# S++ Concept and Application

**Computer software is a mapping to the real world. Object-oriented simulates the world by mapping entities into classes and objects, and completes business modeling through the interaction between objects [1]. S++ conducts the problem from another view. It maps business activities to behaviors to simulate reality, and carry out business scenario modeling through different participants involving in business activities. These are two completely disparate modeling methods. The former first finishes the static model, and then allows the interaction between models; the latter first builds the model of the active scene, and then assigns the actors (participants) to complete the business scene. S++ is a series of inferences based on observed facts (physics is called measurement results or empirical experience). First, a few concepts must be clarified:**

**The concept of action comes from the idea of object-oriented. All object entities can be encapsulated with their own actions to form a complete object. The granularity of the object’s actions can be divided randomly, and there can be no clear business goals and results.**

Behavior is an activity with inherent logic and purpose. Behavior is generally completed by one or more objects (a set of actions), and the behavior must produce business results based on business goals.

Participant is the object entity participating in the behavior. In the S++ method, the behavior exists as the subject, and the objects (including people) all become object participants.

How to distinguish between the two concepts of behavior and action? It can be illustrated by the example of eating. Eating has clear internal logic and purpose (such as lifting hunger), so eating is an act; while opening mouth and chewing may not have a clear purpose (you can eat and also talk or bite things), so it can only be regarded as the action of human entity. In the act of eating, at least two objects, the person and the food, exist. These two objects are called participants in the act of eating.

## Redefining services in SOA

Based on the above concepts, S++ redefines services: a service is the summation of a group of business activities [2]. The following model is an abstract definition of a service:

IMG_256

The capital B, standing for the business activities, has a clear business purpose and inherent business logic, and will eventually produce the certain business results. Business activities are the product of participants and behaviors, and the physical meaning is the behaviors with participations, and the result is the business result targeted by the business goals, expressed by the following model:

IMG_257

Where B represents the goal and result of the business activity, p represents the business participant, and b represents the behavior. Business action(activity?) B is a business goal achieved by business entity p participating in action b. Since neither p nor b can accomplish the business goal independently, it is indispensable to use the form of a product to indicate that (when either p or b is zero, that is when there is no behavior or no entity, the business result B as a product is zero).

* **Distribution Law**

IMG_258

IMG_259

* **Commutative Law**  
  IMG_260

Distribution law and commutative law are the conclusions drawn from the observation and incomplete induction of the real world. It is easy to understand the distribution law because it conforms to the basic method of division; but the commutative law tells us that there can be no order of business activities, which obviously violates intuition. The reason for the inconsistency is that we usually observe business activities in an object-oriented manner. The following S++ method is used to analyze the business activity of hunger relief (eating), and the commutative law can be explained by the independence of the behavior.

**Independency of Behavior**

For business activity B, the hunger relief, there are two participants (p1) and food (p2), and the act(behavior?) b of eating, based on the previous definition we can draw:  
IMG_261  
The literal meaning of this formula is that hunger relief = eating by people + eating by food. If it is analyzed based on the object-oriented method, this is undoubtedly unreliable. How to eat food(eating by food?)? In fact, the actual physical meaning of this formula is: hunger relief = the rise of the human blood sugar + food consumption. S++ can directly refer to the physical nature of the business activities. Once the business activities are decomposed to this granularity, we will find:

* No necessary causal relationship exists between the rise of human blood sugar and food consumption. These two can happen independently.
* If we only focus on the business results, the order of occurrence of the two is not important. ( In fact, for automated services, we actually merely focus on business results. Usually the services are treated as a black box).

The independence of the behavior explains the derivation of the commutative law. Simultaneously, in the process of analyzing the independence of the behavior, it can be found that the business activities cannot be subdivided randomly. After reaching a certain granularity, the business activities that meet the business goals will not be obtained by continuing to split. It means that the business activities are not continuously separable. This characteristic corresponds to the quantum nature of energy. Therefore, S++ defines **the behavior with the smallest particle size as quantum behavior**, and the corresponding **business activity with the smallest particle size is called quantum service**.

## Automated and manual processes

Commutative law is the mapping of **quantum superposition** in the virtual world. It is a human instinct to challenge superposition. We need to realize that business processes can be executed out of order and study the essential difference between automated process and manual process. From the above analysis, we can realize that business activities are composed of quantum services, and for the results, the service can be regarded as a black box. The core of commutative law is based on the manifestation of quantum nature. A business activity can be counted as an uncertain quantum cloud before the observation of its result B from the outside world. The state inside the cloud is completely random, and only collapses to result B at the moment of observation. That is to say, when the result B is observed from the outside world (whether it is observed by human or computer), the observer is actually involved in the entire business process, and the process is differentiated into two stages, before and after observation, by the observer. Therefore, the commutative law losses the function under the observer’s interference.

From the above analysis, it can be concluded that the commutative law merely plays the role in the automated process. The key to distinguish between the automated process and the manual process is not whether anyone involves, but whether the intermediate results of business activities need to be observed during the execution of the process. In other words, any process that needs to read and check intermediate results is a manual process. The commutative law is a important basic law of S++. To make the business activities conform to the commutative law, the observation of the intermediate results of the business process should be completely eliminated. In fact, the unavoidable observations are caused by the technical reasons in the most of the real-world sequential processes. However, these observations are unnecessary in the virtual computer world.

Give an example to illustrate how to eliminate observations in the software, such as the business activity of playing sports in the gym. This business activity can be simplified into two quantum services: “Payment + Play”. In reality, some implementations require to pay first. This is because some business activities cannot be reversed. If you allow playing first without payment, in certain circumstances you may experience a loss because of the inability to pay. In this case, it can be understood that technical means limit the business activities. If the certain technical means can reverse business activites to the initial state, then there is no need to consider whether the two business activities of paying and playing are in order. Fortunately, in the virtual world, almost all behaviors are reversible. All the business activities are nothing more than changes in the data on the storage medium in the computer, so that most of the observations and detections that require intermediate results can be eliminated.

**Quantum entanglement** [3]: quantum superposition is the basis of quantum entanglement in S++. When participants take part in a business scenario together, all participants have aroused the entanglement from a quantum perspective to understand. If entanglement can be achieved, at the moment when any participant in the scene make the changes, all participants will also change synchronously, which is impossible with the current technical means. In S++ implementation, the process of quantum entanglement in the physical world can be simulated by executing all quantum services in the parallel scenes.

# Inference and application of S++ concept

Through knowing the basic concepts of S++, several important inferences can be drawn:

* Business and technology can be separated in services.
* Automation services can be performed out of order.
* Services are invariant in time and space.
* Zero coupling happens between quantum services.
* Services are polymorphic.

These inferences are of great significance to the implementation of S++, and provide a feasible solution to the problems that traditional SOA and microservices cannot solve.

## Separation of business and technology

In real life, almost all actions require the participation of tools, and we usually utilize the technology to optimize our activities. For example, we use tableware to eat, and tableware is a technical participant. With tableware, our eating process is more hygienic and more labor-saving. Another example is that when we drink hot soup, we would use tableware to stir. Stirring will cool the soup quickly to improve the efficiency of our meals. This behavior of stirring is a technical behavior. Through observation, we find that technical behavior and participants have some characteristics:

* Technical participants and behaviors do not change the purpose of business activities. Take an example of the business activity of eliminating hunger. The use of chopsticks, knives and forks, or even direct grasping with hands has no effect on the purpose of eliminating hunger. All technical methods and tools are designed to better help human accomplish business goals, but technology and tools are not the goals.
* Technical participants and behaviors do not change the results of business activities. For any achievable business goal, there must be one or more optimal tools to help people achieve the same business results, and as the technology develops, increasing optional tools can be chosen. No matter any tableware is used in the business activity of eliminating hunger, the results will be the same, and you will not be on short commons due to using chopsticks.
* Technical activities follow the distribution law, which means that technical activities can be performed independently. When the technical behavior and participants are integrated into the service, the mathematical model of the service can be presented as:

IMG_262

IMG_263

IMG_264

Ti represents the items with technology (behavior or participants) in the product. Since these items have no effect on the final result of the business, the value of these items is zero. The separation of technology and business indicates that in a business system, the technical part can be completely independent for development and maintenance, and does not need to be mixed with business code. However, the technical activities cannot comply with the commutative law (because the results of the technical activities are usually observed). For example, encryption and decryption of messages must be performed before or after business execution, the determination of authority must be run before business performance and so on. The separation of technology and business provides great benefits for the development, operation and maintenance of the business systems:

* **What model you build is what you get**: the modeling and implementation of business activities can completely get rid of technical constraints. The traditional system development model, experiencing the process of requirement, analysis, design, coding, testing and then launching online, has transformed to two independent and unrelated processes: modeling process from business experts and meeting technical requirements by technical personnel.
* **Zero coding for business modeling**: for services that conform to automated processes, the modeling process would not focus on technical implementation and any branching logic, but on defining participants and behaviors and making a list. Therefore, the coding is not needed.
* **Consistency of the service connotation and diversity of the service extension**: the business part B in the service reflects the service connotation and the technical part T reflects the service extension. The diversity of business is reflected in the richness of changes of T, such as the diversity of payment services. As long as the technology makes a slight change, it may lead to earth-shaking changes in the entire service, but no matter how the service changes in form, its business connotation is stable and unchangeable, which brings huge benefits to the development of business systems based on S++. In the process of the business modeling of the services, if you merely need to pay attention to the meaning of the service, it will be undoubtedly much simpler for the modelers; similarly, the technical personnel only need to focus on improving the technical capabilities of the system, instead of understanding the business rules.

## Out-of-order execution/parallel operation

The greatest benefit based on the commutative law is parallel operation. Automated services can implement parallel operations at the quantum service level. An unsolvable problem of the microservice architecture is that as the granularity of the service decreases, the performance of the service composition drops down sharply. Assuming that the time Ts, presented the time of running a service, includes: communication time Tc, database transactions time Td, and consuming time of the service T0~Tn, and then the time consumed by service calls in different architectural forms can be expressed using a mathematical model.

* For the traditional SOA services with high cohesion and loose coupling, the steps required by business activities are generally finished in a database transaction, so the time it takes to call is expressed as follows:

IMG_265

* For microservices, since the service granularity is much smaller than SOA. We assume that in the extreme cases, all steps in SOA are split out and called separately, and then we have:

IMG_266

* For S++, the benefits of parallel operation are:

IMG_267

It can be concluded from the above three mathematical models that if the SOA service is used as the benchmark, the performance of microservices will consume n-1 more times of communication and database transaction times as the granularity decreases, while S++ can speed up to operation with the decreasing of granularity.

For microservices, due to the gradual reduction of service granularity, no microservice can independently provide the complete support to business scenario. Therefore, the combination of services cannot be avoided to be utilized in large quantities. If we don’t carry out S++ for modeling, microservices cannot tackle the performance problems caused by reducing the granularity.

## Time and space invariance of service/service de-versioning

To design the business services with the mind of the technical personnel, the version of the service must exist intuitively. However, from the perspective of separation of business and technology, no matter how plentiful the extension of the service is, there is no change in the business connotation after stripping off the technical part. S++ defines this characteristic of service as **the time and space invariance**. The time and space invariance of services is consistent with the observation of the objective world. For example, the business activity of car maintenance has been in advent since ancient time, and no matter how the era changes, the goal and result of this business are still: bad cars go in and good cars come out.

In fact, relying on the intuition to apply the version characteristics of objects in the process of establishing a service model has fatal consequences, especially for achieving business requirements through composition. If service A is called by a composite service S, the service S should maintain two versions when service A needs to maintain two versions. This seemingly simple question encountered a huge challenge after the service S is combined with multiple services (such as A, B, C). If it happens that A, B, and C all need to maintain two versions respectively, does the service S need to maintain 2x2x2=8 versions to cope with the combination of different versions? Obviously, this is impossible. The version of service S++ provides:

* The connotation of the service does not allow having versions.
* Any modification to the connotation of the service will inevitably result in a new service.
* The version of the service is only allowed in the external definition of the service.
* The combination of services can only call the connotation of the service, and it is not allowed to directly call specific extensions (implementation of services).

## Zero-coupling between quantum services

If a quantum service A needs to call another service B, then according to the definition we have:  
IMG_268  
A is not a quantum service, so there is no calling relationship between quantum services, which means no coupling.

The description of zero-coupling of quantum services is:

* Business activities must be realized by the combination of quantum services.
* Every coupled business system can be further split into two parts: quantum services and business portfolio

The traditional microservice architecture has a serious problem. As the granularity of the services is refined, the complexity of service systems increases exponentially. The mutual call between services causes the increasing transaction links with the number of services, which will lead to a lot of problems:

* The reduced maintainability of the system: any modification on the service will cause the associated modification, while the number of the services is still huge.
* The complexity of the transaction link is uncontrollable, resulting in a extreme dependence on the monitoring system.
* The complexity of the monitoring system has risen extraordinarily, and the demand for resources even exceeds the capacity of the business system. For instance, only one person is working and a group of people are watching and supervising him or her.

In this case, to simplify the transaction link and system complexity, S++ provides:

* No service can include both the specific business and the calls to other services.
* Quantum services can only be called by purely composite services.
* The composite service cannot call each other. The composite services can only be modeled by listing the participants and behaviors.

## Service polymorphism

In the object-oriented method, the objects are the business subjects, the types of objects cannot be enumerated, and the instances of the objects cannot be counted. The complexity of system modeling relies on describing the complicated object-relational models. For S++, business behavior is the main part and the behavior patterns are too few compared to the objects, and can be enumerated. How does S++ build a variety of business services using the extremely limited behaviors? It is the participants who determine the business goals and results of the service. For example, for the business behavior of eating, when the participant is a diner, the business goal is to relieve hunger; if the participant is a guest at a business banquet, the business goal becomes social relations; if the participant is an athlete who fought eating contest, and then the business goal is to attain the medals.

**The same business behavior leads to the different business goals and results due to different participants. This has shown the polymorphism of the service.** The polymorphism leads to a large number of business services in S++ have similarity to each other.

Service polymorphism can improve the maintainability of business processes and remove business branches in traditional business processes. The traditional processes need to determine the type of business based on the contents of business and subsequent service calls; in S++, the decision and scheduling of business branches are automatically performed by the system and the different business services are invoked according to the different participants. For instance, for the payment business of the bank, the traditional business process needs to decide the payment category field, and the content of the field will be identified to conduct the payment of telephone bills, water bills, etc. If you want to add a payment category, you must modify the business process. It remarkably reduces the maintainability of the system; in the business process of S++, we do not need to make the business branch identifications. In the modeling process, only the abstract payment service needs to be called. After the abstract participants are replaced by the actual participants during the operation process, the system will automatically consummate business branch function.

The most principal difference between the S++ service polymorphism and the object-oriented polymorphism is that object-oriented polymorphism must be conducted on the client side, while S++ polymorphism can be implemented on the server side. This difference is mainly reflected in the different coupling degree of the system. For the object-oriented polymorphism, the client must couple all known implementation classes and at the very least it should be the interface level of the coupling (such as remote objects). Nevertheless, the client of S++ can never know how many specific services are implemented. The service polymorphism is decided by the server according to the received message. Without(except?) the abstract services, no coupling would happen.

## The category of services

Services in S++ are divided into two categories: technology class and business class. Each category is further subdivided.

### Technology class

Based on the definition, all actions(behaviors?) and participants that have no impact on business goals and results are classified as the technical category. Therefore, technical services can be easily distinguished [].

Calibration services

The calibration services are mainly used for data inspection and the verification of the input information of the service to prevent the dirty data from polluting the service system.

Conversion services

Conversion services are mainly used to convert input data in different formats into the data with the format specified by the service system, or reverse the conversion.

Security services

Security services are primarily performed for encryption and decryption of sensitive data, anti-tampering of messages, system authentication, authority control and other measures to ensure transaction security.

Error handling services

Error handling services is to intercept business and technical errors that occur during service execution through an error interceptor, so that to separate error handling from business processes and avoid the interference with the business.

### Business class

Business services include five major categories according to behavior patterns [4][5]:

Entity maintenance

Even though S++ service and object-oriented service have similarities in the maintenance of entities, there are also major differences between them. The biggest difference is that the entity class of S++ only includes the natural classifications and attributes of the entity. For example, the entity of human can generate a large number of classifications from different dimensions in object-oriented service, such as classification by occupation, age, career, etc.; but in S++ service, the only attribute classified by nature is the category of human, so all human entities are identified as the class of natural persons. The social attributes of human are expressed through the results of contracting behaviors. Natural entities become a participant through contracting. For example, natural persons contract with the company to become employees. Usually most services are directed to participants rather than natural entities.

Contract Signing

Contract signing is the most common behavior in social activities. Generally, contracts are divided into identification contracts and executable contracts. Identity recognition contracts can be signed between natural entities or mixed between natural entities and participants. The result of identity recognition contracts is to generate new types of participants in order to meet certain business scenarios. Executable contracts must be signed between participants. Natural entities must be attached to social attributes before they can take part in the signing of an executable contract. An executable contract usually has multiple execution states and results. For instance, an order is an executable contract. The order can be modified, paid, canceled, dispatched, closed, evaluated, etc. The change of the contract status in S++ service actually produces a new contract.

Quantum service class (generated during contract execution)

Contract execution services are applied to execute contract state changes and contract participant attribute changes. Changes in the contract state will produce the new contracts, and only changes in the attributes of participants will truly affect the physical world. Normally, the changing physical quantity in the business environment will only be shown in the form of numbers, such as the inventory quantity, account balance, etc., so it seems that the quantum services only have two circumstances of increasing and decreasing.

Query class

Query services are used to query entities, participants, and contract execution results.

Computing class

The computing services should be a part of other business services in a strict sense. For example, the calculation rate is a part of the charging service, and the charge execution can be conducted after the rate charge rate is calculated. The purpose of separating computing services from business services is to improve the reusability of computing processes. Therefore, computing services are more closed to technical services. In the dealing with the process, the computing class is similar to the technical class. The business calculation work needs to be completed according to the different participants before the automatic process is initiated.

## Others

### Service Security

Quantum services in S++ service do not have any coupling calls with each other, and all calls come from the combined process of the business scenario, so S++ is easier to obtain at a low cost in security than traditional SOA and microservices. For quantum services, only a simple IP whitelist is required to ensure the legality of access, because only a limited number of process control nodes can access quantum services. As long as the traditional security measures are applied at the entrance of the combined process, the security of the entire system can be ensured.

### Data storage method

S++ supports the most thorough distributed storage (a centralized storage structure), and the data storage layer designed for the characteristics of quantum services can exploit the advantages of S++ to the full. In the S++ model, the relationship between objects is greatly weakened, and the types of data tables are limited as well. In the current design, three types of data tables include entity tables, contract tables, and quantum data tables. Compared with the traditional relational database, it has the following characteristics:

1. Database services exist in the form of discrete microservices. Centralized database management procedures, similar to relational databases, are not required and there is no strong dependency between services.
2. The database service mainly provides three primary service functions: entity maintenance, business pipeline(log?) maintenance and quantum service. The traditional function of retrieving on content is independent as a third-party indexing service.
3. Database records can be accessed through array subscripting, especially the data that quantum services need to change, and high-speed random storage devices can be used independently to enhance the performance much higher than traditional relational libraries.
4. Except for quantum data, all other data are modified by sequential addition, so you can promote write efficiency by adopting cache.

**Reference**

1. Suhardi, S., Doss, R., & Yustianto, P. (2015). Service engineering based on service oriented architecture methodology. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *13*(4), 1466-1477.

2. Carter, S. (2007). The role of business process management in SOA. *Information Management*, *17*(5), 30.

3. Gyongyosi, L., & Imre, S. (2018). Entanglement availability differentiation service for the quantum internet. *Scientific reports*, *8*(1), 1-8.

4. Mahmoud, T., & Gómez, J. M. (2008, April). Integration of semantic web services principles in soa to solve eai and erp scenarios; towards semantic service oriented architecture. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications* (pp. 1-6). IEEE.

5. Muthusamy, V., Jacobsen, H. A., Chau, T., Chan, A., & Coulthard, P. (2009, November). SLA-driven business process management in SOA. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research* (pp. 86-100).