The Micro-service Architecture Design Research of Financial Trading System based on Domain Engineering

Xing Sheng [a], Shuangshuang Hu[b], Yihui Lu[c]

CFETS Information Technology (Shanghai) Co., Ltd, China

[a]shengxing_zh@chinamoney.com.cn, [b]hushuangshuang_zh@chinamoney.com.cn,

[c]luyihui_zh@chinamoney.com.cn

Abstract. The purpose of this article is to propose a domain engineering-based micro-service reference architecture for financial trading platform, which can solve the problems of high complexity and high maintenance cost of a real large-scale financial trading system and try to provide a general solution for similar scenarios in the future. Starting from the concept of domain engineering and micro-service and the relationship between them, this article briefly introduces the background and pain points of the actual large-scale financial transaction system, emphatically expounds the principles and methods of domain analysis and micro-service disassembly for the legacy system, and puts forward a reference architecture with future reuse significance. The corresponding suggestions are also given for how the new micro-services will coexist with the existing legacy systems for a long time. In order to promote the landing of domain engineering and micro-service, besides theoretical research and necessary practice, enterprises also need to change the traditional management mode and introduce DevOps and other mechanisms, so that business processes and technologies can support the rapid development of business.

Keywords: Domain Engineering, Micro-service, Legacy System, Reference Architecture.

1. Introduction

With the development of information technology and the complication of enterprise business, the application of the system is becoming more and more complex, the code size is getting larger and larger, the maintenance and the costs of updating and difficulties are continuously improved. The development of technology and the expansion of business have brought new challenges to the deployment of the system and put forward new requirements for the efficiency and expansion of the enterprise application architecture [1].

Componentization [2] is an important tool for solving complexity problems. It divides a huge application into multiple components, each of which performs independent functions and works together. The componentized implementation of the original system is using the library, which can separate the development, compilation and testing, but cannot be completely divided according to the business function, the code of each business function is mixed together, the difficulty of debugging and deployment is larger. When a library or component is modified, you need to restart the application.

With the emergence of micro-service architecture [3-6] and domain engineering [7-10] and its successful application at home and abroad, it provides new ideas and implementation solutions for the transformation of a single old system. In this paper, we will analyze the application system with domain engineering modeling, design the framework model with the common domain requirements, and provide theoretical basis for component division. Componentization through micro-services, to split applications into a single service, and reduce coupling between components.

2. Overview of Domain Engineering and Micro-service

2.1 Domain Engineering

Domain is the area covered by a set of application systems with similar requirements and functions. Domain Engineering refers to the activity of collecting, organizing and preserving experience in a reusable form when constructing a new system or some parts of a system in a specific domain, and providing an adequate way to reuse these resources. It analyses lots of systems in the same field and draws their common domain requirements, to design a framework model that meets the requirements. Finally, it develops and organizes reusable components based on the above framework model. In this way, when developing new applications in the same field, the requirements of new applications can be determined according to the domain model and the specific domain. Domain software architecture can be used to generate the design of new applications, on which basis, reusable components are selected for assembly, thus forming a new system.

From the above definition, we can conclude that domain engineering can be roughly divided into three steps, domain analysis, domain design and domain implementation.

Domain analysis is concerned with how to design a set of accurate, concise and correct real-world models to understand requirements in depth by creating models. The key point is to abstract the core features and put details into specific design processes.

The model is the selective simplification and purposeful structuring of knowledge. Domain model is not only the knowledge in the domain expert's mind, but also the abstract knowledge which is strictly organized and carefully selected. In the field of software development and design by domain engineering method, the role of domain models cannot be ignored. First, the core of the model and design interacts. The close relationship between the model and the implementation ensures that the analysis we make in the model can be transformed into the final product. Secondly, model is the center of communication languages used by team members. Domain model can be used to promote the communication between developers and domain experts. In addition, the model is the concentrated knowledge, and the model is the way to organize domain knowledge and to distinguish the most important elements. Many complex projects are actually trying to use domain models, which are worthless if the entire programming or core parts do not correspond to the domain model. So, we must strictly guarantee the consistency between model and design within a certain range. Object oriented method provides modeling support for this model, and also provides a way to implement the model constructions.

Domain design is the second stage of domain engineering, of which the main goal is to develop a corresponding design model for the problem domain and to express it explicitly.

In the process of preliminary domain design, four parts of OOD [11] model should be designed, namely problem domain, human-computer interaction, control interfaces and data interfaces. The designs of these four parts have no certain order in time, and they can be interpolated according to the actual situation. The information sources of this activity mainly include domain requirements definitions, object-oriented analysis models and design of existing systems in this domain. Systems in the same field are often similar in terms of implementation. The task of preliminary domain design is to establish a basic OOD model and grasp the commonalities of these implementations. Since existing systems are developed in their own specific environments, there may be two or more different solutions to the same problem, in which situation if there is a solution that can adapt to the variability of the systems in the field, and there are no obvious side effects (such as reduced efficiency or impact on other parts of the design) when applied to various situations, then this solution should be adopted. Otherwise, multiple solutions should be chosen to make them suitable separately.

The main activity of domain implementation is to use appropriate technology and language to implement product architecture and components in domain design. Domain Specific Language (DSL) [12] is a specific problem-oriented language. After implementing domain architecture and components, DSL can be used to generate specific software products. Which can be specific source programs or compiled software modules.

In the fourth part of this paper, we will elaborate on the whole process from domain analysis to domain realization based on specific examples.

2.2 Micro-service

The so-called micro-service is to divide the functions in the application as fine as possible and treat each small function as a separate service. These tiny services usually use HTTP API to communicate, and they focus on specific business functions, have strong module boundaries, and can be deployed independently.

When we talk about micro-service, people always compare it with SOA (Service Oriented Architecture). Strictly speaking, we can consider micro-service as a subset of SOA. In the late 1990s, SOA [13] first proposed the idea of using low-coupling and service-oriented processes in software architecture design. However, in the architecture of SOA, the complex ESB enterprise service bus [14] is still in a very important position. The architecture of the whole system has not been fully componentized and service-oriented, and its learning and using threshold is still high.

The idea of micro-service architecture originates from the horizontal and vertical cutting and splitting of business functions and modules in project design. In 2012, the structure of micro-service was proposed, and since then many design cases have come into being. In the following years, Amazon, Uber and other enterprises carried out their own practices and all of them achieved successes. Micro-service emphasizes completely component-based and service-oriented. All micro-services are independent, and they are exposed to callers in the form of RESTAPI externally.

Micro-service is essentially a design style of software architecture. The whole software service architecture is composed of multiple micro-services. It does not have certain rules and needs to be designed according to business requirements. The characteristics of micro-service architecture can be roughly summarized as following three points.

Complexity controllable: Micro-service can decompose applications into manageable branches or services. Through the micro-service architecture model, complex functions can be presented in a modular way, which makes it easier to develop and maintain a single service.

Flexible and scalable: The micro-service architecture enables each service to expand independently, and each service can add or subtract functions independently, making the whole system very flexible.

Independent deployment: Micro-services have independent running processes, so each micro-service can also be deployed independently. Using the same deployment environment enables batch rapid deployment of micro-services.

Compared with the traditional single application architecture, micro-service has obvious advantages in many aspects.

2.2.1 Heterogeneity

Problems are often concrete, and the solutions should be targeted. The heterogeneity of micro-service can help developers select different technical solutions according to different business characteristics and solve specific business problems pertinently.

As for the heterogeneity of micro-service, we will elaborate in detail in Chapter 4 how we introduce micro-service into legacy systems to make new micro-services coexist with existing systems for a long time.

2.2.2 Independent Test and Deployment

Under the micro-service architecture, packaging, testing and deployment of different services are completely independent. From this point of view, the cost and risk of code modification, testing, packaging and deployment are much lower than that of single application architecture.

2.2.3 On-demand Expansion

Due to the limitation of single process, single application architecture can only be extended horizontally based on the whole system, and it cannot be extended on demand for a specific functional module. The micro-service architecture can perfectly solve the scalability problem. The system can be extended as required.

2.2.4 Error Isolation

The micro-service architecture can also enhance the isolation of errors or faults. For example, if a certain service leaks memory, it will only affect itself, and other services can continue working properly. In contrast, if an unqualified component is abnormal in a single application architecture, it may drag down the whole system.

2.2.5 Full Functionalization of the Team

The traditional development model usually takes technology as the unit of division of labor, such as UI team, server team and database team. Micro-service advocates a division of labor according to services. Team members need all the skills to design, develop, test and deploy services.

In summary, the advantages of the micro-service architecture are obvious. However, no software architecture is perfect. Micro-service architecture also has its shortcoming.

Firstly, the design of micro-service architecture does not make all operations service-oriented and componentized. For example, some underlying operations at the database level are not recommended to be service-oriented. The architecture designer needs to make a reasonable division according to the specific circumstances of the business. In addition, the purpose of the micro-service architecture is to solve the efficiency problems in software development and iteration. Because the service functions that should be used internally are exposed in the form of API, it is necessary to add several times of internal data transmissions based on HTTP protocol in one external data service process of software platform. Although the transmissions are completed in the intranet, the calls to the function interfaces in memory are much slower, which is a big challenge for the design and deployment of software platform server when the amount of service increases in later periods. Finally, due to the exposure of APIs in the intranet, designers also need to authenticate key micro-services to ensure that some sensitive micro-services are not abused.

We briefly introduced the concepts and implementation principles of domain engineering and micro-service. What is the relationship between domain engineering and micro-service? How can we apply domain engineering modeling ideas and methods to provide theoretical basis and technical support for the separation of micro-services?

As we all know, the key to designing micro applications and micro-services is to decompose the architecture. According to the architecture decomposition model, software architecture can be decomposed from four dimensions, business domain, function domain, technology domain and public domain. The application architecture based on micro-service mainly involves three dimensions, business domain decomposition, functional domain decomposition and technology domain decomposition. Business domain decomposition decomposes applications into subsystems, functional domain decomposition decomposes subsystems into sub-modules, and technology domain decomposition divides sub-modules into frontend micro-applications, backend micro-services and sub-databases. These three decompositions are iterative processes, from coarse to fine and from fuzziness to clarity. The division of business domain and function domain can be based on the idea of domain engineering to analyze and design the domain of single application, so as to lay the foundation for the subsequent micro-service transformation.

In the follow-up part of this article, we will focus on the domain engineering-based micro-service decomposition, while we will not focus on the technical framework and details of micro-service implementation.

3. The Introduction of RMB Trading System

The Interbank RMB Trading System (hereinafter referred to as the RMB Trading System), which is currently in operation, is launched in 2009. It is a complex and huge trading system with the following problems:

High complexity: The current RMB Trading System has millions of lines of codes, including a lot of intersystem interfaces and functional modules. Some functional modules have unclear dependencies or blurred boundaries, and the code style and quality are uneven, resulting in incomplete assessments may be made when making new requirements or modifying defects, even a small change may result in unpredictable defects.

Poor reliability: Due to the high coupling of each functional module, small defects in one functional module may cause other functional modules to be unavailable, and the system may not be used.

Legacy technical debts: The RMB Trading System has been in operation for more than ten years. Over the time, changes in demands and the developers change, the technical debts of the entire system have accumulated more and more. Many developers believe that there is no defect to be modified, the revised idea has made the current system more difficult in design and code modification. If encounter new requirements that are significantly different from existing implementations, the cost of modifying the code is very high, resulting in having to sacrifice some of the requirements so that the system changes are within the controllable range, causing the business team to be dissatisfied with the development team.

Difficulties in deployment: With the increment of codes, the construction and deployment time of the RMB Trading System has also increased. At present, it takes about half an hour to compile all in the background, and it takes about one hour for the client to compile once. Every time the client needs change or defect repair needs to re-release the client, this full-scale release method takes a long time and has a large impact range, which inevitably leads to higher risks, so that it cannot respond to business needs quickly.

Limited expansion capability: At present, the RMB Trading System cannot support horizontal expansion according to business needs, and business processing can easily reach bottlenecks. In addition to logical optimization, it is difficult to expand from the architectural level, resulting in a worse user experience.

As mentioned earlier, micro-services are small services that consist of a single application. They have their own processes and lightweight processing. The services are designed according to business functions, deployed in a fully automated manner, and communicated with other services using the HTTP API. At the same time, the service uses the smallest scale of centralized management (such as Docker) capabilities, services can be implemented in different programming languages and databases. The complex and controllable, flexible expansion and independent deployment of micro-services can solve various problems in the current RMB Trading System.

4. The Solution based on Domain Engineering and Micro-service

4.1 The Overall Principles and Methodology for Reforming

To retrofit legacy systems based on the architectural philosophy of micro-services, we can follow these steps:

Firstly, analyze the functional modules included in the legacy system, extract the common parts, and subdivide each functional module into small functional points as small as possible;

Secondly, extract the various function points obtained by analysis and extract the independent modules;

Third, strip out the business data, try to make the business data of each micro service independent of each other, isolated from each other without affecting;

Fourth, analyze and determine all the functional points that need to be implemented, and determine the business process and development technology selection of each service;

Fifth, according to the technical characteristics of micro-services and the business needs of large platforms for summary design and detailed design;

Sixth, rapid development and testing of code;

Seventh, the container is used to encapsulate the resource environment used by the micro-services. The same container can be directly deployed.

4.2 The Typical Reference Architectures for Legacy Financial Trade System based on Micro-Service Transformation

4.2.1 Aggregator Micro-service Design

The aggregator [15] calls multiple services to implement the functionality required by the application. It can be a simple web page that processes the retrieved data. It can also be a higher-level combined micro-service that adds business logic to the retrieved data and then publishes it into a new micro-service, which is consistent with the DRY principle. In addition, each service has its own cache and database. If the aggregator is a composite service, then it also has its own cache and database. The aggregator can be independently expanded along the X and Z axes.

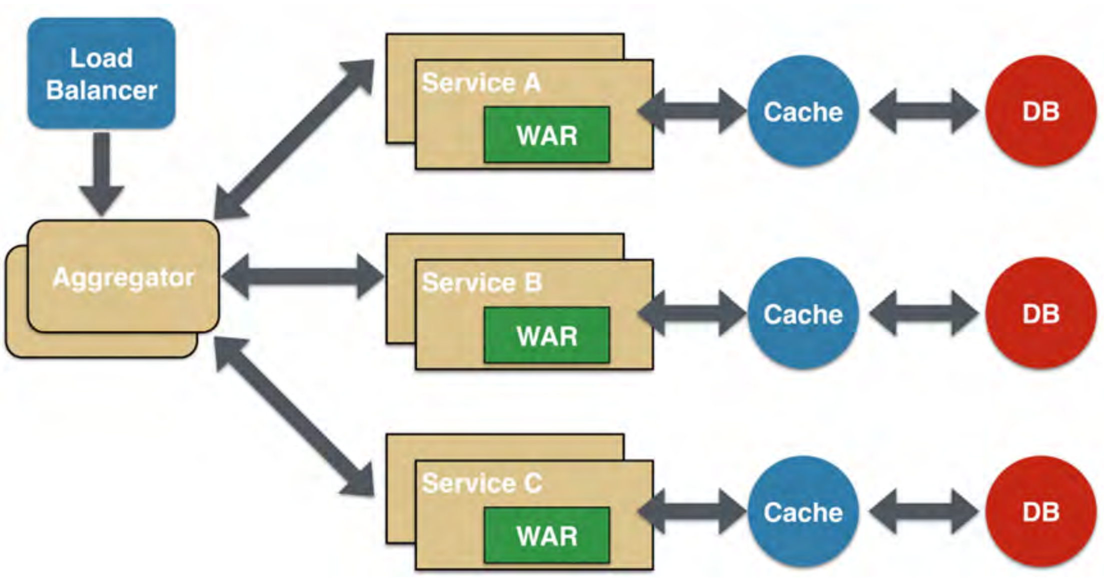

Fig.1 Aggregator micro-service design

4.2.2 Proxy Micro-service Design

Proxy micro-service design [16] is a variant of aggregator micro-services design. In this case, the client does not aggregate data, but calls different micro-services based on the difference in business requirements. The proxy can only delegate requests or perform data conversion tasks.

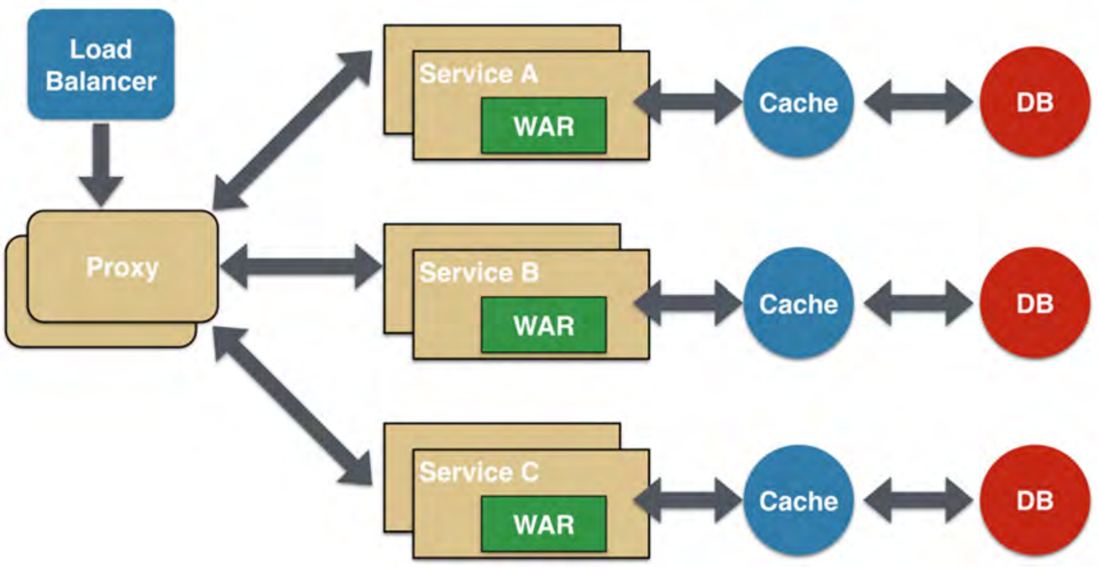

Fig.2 Proxy micro-service design

4.2.3 Chained Micro-service Design

Service A will communicate with Service B upon receiving the request. Similarly, Service B will communicate with Service C. All services use synchronous messaging. The client will block until the entire chained call is completed. Therefore, the service call chain should not be too long to avoid waiting for the client for a long time [17].

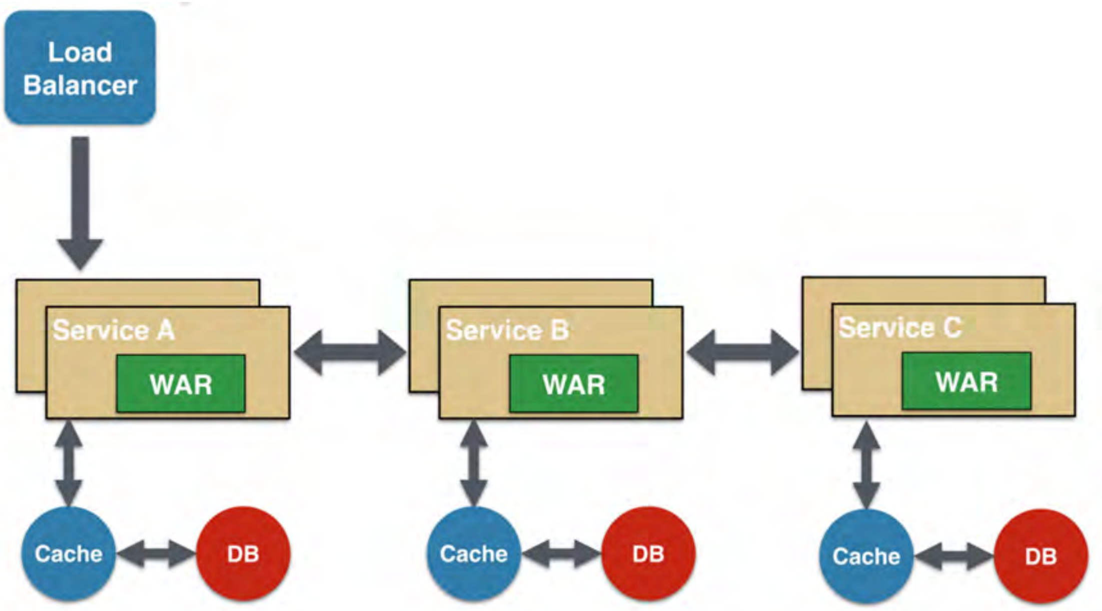

Fig.3 Chained micro-service design

4.2.4 Asynchronous Messaging Micro-Service Design

Although the REST design pattern is very popular, it is synchronous and can cause blocking. Therefore, some micro-service-based architectures may choose to use message queues instead of REST requests/responses [15].

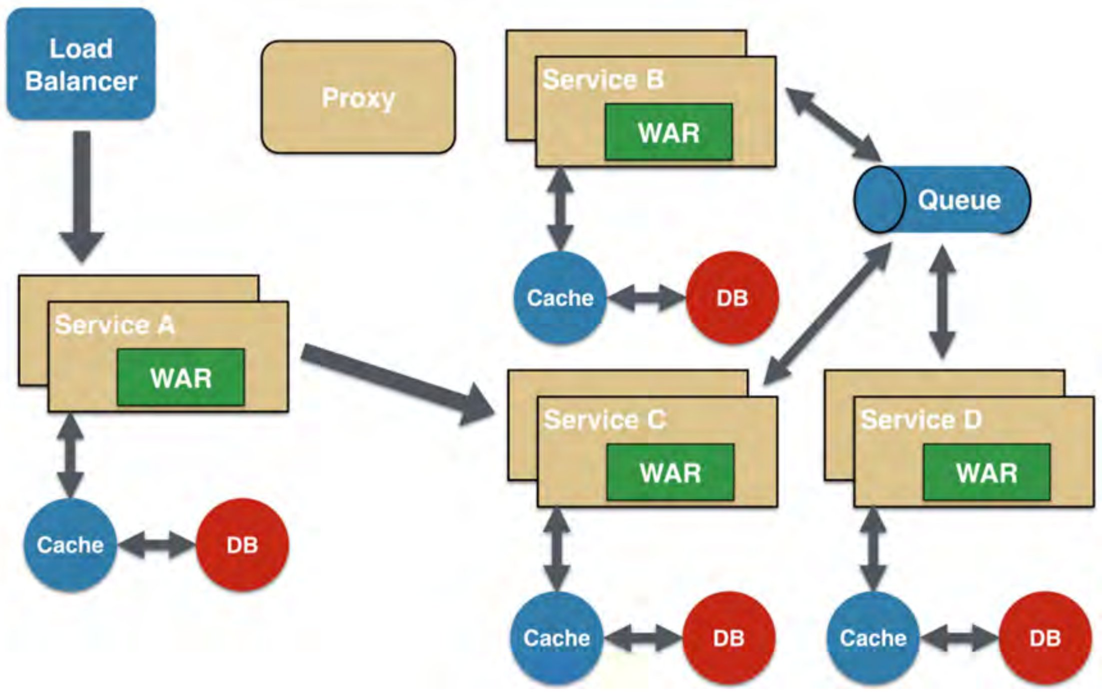

Fig.4 Asynchronous messaging micro-service design

4.3 Micro-service Dismantling Principle

Based on the principles and methods of the overall transformation of a typical financial transaction legacy system, the principles used in the use of micro-services disassembly are as follows:

Single responsibility, to achieve high cohesion and low coupling, the micro-services should define moderate granularity to avoid circular dependence and two-way dependence. To be based on domain analysis and design, each service has a clear scope of responsibilities and boundaries.

Evolutionary split can adapt to the rapid iteration of the version. The consistent interface and data flow can eliminate duplicate data, create a clear recording system and a consistent integration interface.

Cut with the business model and fully consider the independence and professionalism of the business. Business needs need to be considered when designing and decoupling the architecture. In the service dismantling, taking the business perspective first, fully considering the independence and professionalism of the business, reasonably divide the boundaries according to the business functions of the service.

Consider the team structure, insist on dismantling the benefits, and not increase the maintenance cost for disassembly. The measure of the dismantling benefit is that the system maintenance cost after disassembly is lower than the maintenance cost of the system before disassembly. The so-called maintenance costs include manpower, material resources and time. Considering that the old and new systems need to be run in parallel during the transformation, it is not possible to increase the manpower and the increase in the capacity requirements of the personnel due to the dismantling of the micro-services, which may lead to a decrease in the proportion of input and output.

Consider compatibility, and the new and old systems will smoothly transition. When using the new technology, the micro-service transformation of the function is performed, the interface should be as transparent as possible, and the API changes should be transparent to the user.

In short, when we are dismantling micro-services, we must adhere to the principles of business-oriented, road-to-simplification, and divide and conquer, and fully consider single responsibility, service granularity, business needs, dismantling benefits, and version compatibility, not just from a technical point of view and is assemble the service into many small modules.

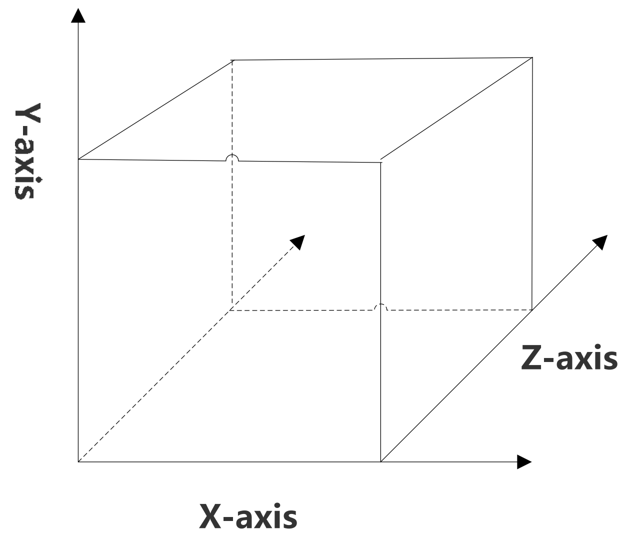

Fig.5 AKF

Based on the above disassembly principle, we adopt the AKF extension principle [18]. The AKF Extended Cube (The Art of Scalability) is the three dimensions of the application extension of the abstract summary of AKF's technical experts. In theory, according to this extended mode, the single system can be expanded indefinitely and is highly scalable.

X-axis: refers to horizontal replication, in short, single-system multi-instance operation, and the cluster adopts load balancing mode.

Z-axis: refers to similar data partitioning. When a trading product belongs to a high-frequency trading product, the data is partitioned according to the transaction heat, and the data processing volume of each partition is relatively balanced, thereby ensuring the real-time data processing.

Y-axis: refers to the split mode of the micro-service, based on different business splits.

The advantages of using this splitting principle are: 1. Low coupling, high cohesion: one service is only used to complete a single independent function; 2. It can achieve maintenance by team structure and small-scale team to achieve fast iteration. The service characteristics of each service are different, independently maintained, and support unlimited expansion.

4.4 Domain Entity and Object Definition, System Splitting

Based on the old financial transaction legacy system, domain analysis is carried out from the perspective of demand. First, we need to define domain entities and domain objects. A domain entity is a domain concept that needs to be uniquely identified in the domain. In other words, it refers to an object that is usually uniquely identified and distinguished in the business and needs to be continuously tracked for it. An object is an entity if it maintains continuity throughout its lifecycle and is independent of its attributes (even if these attributes are important to system users). A domain object can be thought of as an attribute of an entity. The fundamental difference between an entity and an object is that the object only needs to know what it is. The entity not only needs to know what it is, but also needs to know which one it is.

A domain consists of multiple subdomains, each of which corresponds to a different component of the business. We divide the subdomains into three categories:

Core sub-domain: The core distinguishing point of the business, the most valuable part of the application, needs to reflect the unique competitiveness of the system.

Universal sub-domain: not specific to the business or does not have personalized appeals.

Support subdomain: It can be understood as different from the existence of the above two subdomains.

Based on the above definition of some concepts in the field, we can design the transaction model as follows:

Core sub-domain: quotation, transaction, order;

Universal subdomain: user login, permission verification;

Support subdomain: calculation, market;

4.5 Coexistence of New Micro-services and Existing Systems

In the process of disassembly, we can ensure the normal operation of the system and maximize the smooth transition between the old and new architectures according to the following principles.

4.5.1 Proceed in an Orderly Way and Step by Step

Through the way of "function modules stripping + a small amount of interface code modifications", the gradual stripping of system functions is realized. The order of stripping follows the principle of "easy before difficult, from outside to inside". Starting with the function modules of system peripheral and supporting auxiliary functions, it gradually goes deep into the core business function modules. In our case, we give priority to the transformation of the maintenance and query module of the trading system, because these functional modules are relatively independent, and even if the transformations are difficult or functional problems occur, it will not have a significant impact on the core functions of the system.

4.5.2 Coexistence of New and Old

Based on the idea of "step by step", we abandon the previous system upgrade method of "marking a certain time point and switching between old and new systems" and adopt the concept of "continuous release" to maintain the coexistence of old and new systems in the process of transformation. That is to say, after each new service is stripped off the line, the frontend proxy is used to transfer the relevant functional requests to achieve transparent service switching to users.

4.5.3 Data Redundancy and Synchronization

Under the micro-service architecture, each service should have its own independent data storage structure. Therefore, in the process of service disassembly, on one hand, data need to be decoupled to achieve functional decoupling, and on the other, data redundancy can be used to deal with the data that is difficult to decouple completely. According to the real-time requirements of data, data synchronization tools can be used to synchronize relevant data to the new service node.

5. Summary and the Future Work

Starting from the concepts of domain engineering and micro-service, this article proposes a design idea of micro-service based on domain engineering and tries to apply it to the transformation of legacy system of monolithic architecture. It is relatively easy to create a system with a micro-service architecture from scratch, because there are no constraints in the existing system. How to dismantle and gradually transform the legacy system and how to ensure the smooth coexistence of the new micro-services and legacy system to the greatest extent is the focus of this article. It is also a valuable wealth that we think can provide guidance and reference for the follow-up similar legacy system transformations. In the fourth chapter of this article, we introduce a typical micro-service reference architecture of financial transaction legacy system, which also provides more ideas for the future design of similar business systems.

From the technical point of view, this article explores and demonstrates the application architecture based on the micro-service architecture and focuses on the methodology and reference architecture of the existing system transformation. At present, the micro-service transformation of the financial transaction legacy system introduced in this article is still in progress. We will continue to summarize and refine the transformation, analyze the problems encountered in the micro-service implementation level, and research and implement the development, deployment, routing and monitoring tools needed for micro-applications and micro-services.

In addition, the micro-service architecture brings not only technological changes, but also changes in management methods. To better apply the micro-service architecture, enterprises need to change the traditional management mode, among which DevOps [19] is a management concept that complements the micro-service technology. By introducing DevOps mechanism, the application deployment package constructed in the development phase is made into a container image and serves as a delivery through the subsequent testing and implementation phases. Due to the portability of the container, the same application operating environment can be cloned quickly in the testing and production environments to realize the integration of development, operation and maintenance. Like the enterprise organization structure and delivery process, we need to carry out drastic reform of the existing mechanism. Once the process and technology innovations are completed, the cost of future system construction will be greatly reduced, and system quality will be greatly improved. The ability of the enterprise to embrace changes in business will also gain a qualitative leap.

References

[1]. Will Tracz, Lou Coglianese, Patrick Young. A domain-specific software architecture engineering process outline. ACM SIGSOFT Software Engineering Notes. Vol. 18 (1993) No. 2, p. 40-49.

[2]. Shimin Li, Ladan Tahvildari, et al. A Service-Oriented Componentization Framework for Java Software Systems. 2006 13th Working Conference on Reverse Engineering. Benevento, Italy, 23-27 Oct., p. 1-10.

[3]. Richardson C. Microservice architecture patterns and best practices [J]. URL: http:// microservices. io/index. html [accessed: 2016-02-12], 2016.

[4]. Le V D, Neff M M, Stewart R V, et al. Microservice-based architecture for the nrdc[C]//Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on. IEEE, 2015: 1659-1664.

[5]. Butzin B, Golatowski F, Timmermann D. Microservices approach for the internet of things[C]//Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on. IEEE, 2016: 1-6.

[6]. Braun E, Schlachter T, Düpmeier C, et al. A generic microservice architecture for environmental data management[C]//Environmental Software Systems. Computer Science for Environmental Protection: 12th IFIP WG 5.11 International Symposium, ISESS 2017, Zadar, Croatia, May 10-12, 2017, Proceedings 12. Springer International Publishing, 2017: 383-394.

[7]. Falbo R A, Guizzardi G, Duarte K C. An ontological approach to domain engineering[C]//Proceedings of the 14th international conference on Software engineering and knowledge engineering. ACM, 2002: 351-358.

[8]. Shur V Y, Rumyantsev E L, Ndcolaeva E V, et al. Recent achievements in domain engineering in lithium niobate and lithium tantalate[J]. Ferroelectrics, 2001, 257(1): 191-202.

[9]. Bjørner D. Domain engineering[M]//Formal Methods: State of the Art and New Directions. Springer, London, 2010: 1-41.

[10]. Aharoni A, Reinhartz-Berger I. A domain engineering approach for situational method engineering[C]//International Conference on Conceptual Modeling. Springer, Berlin, Heidelberg, 2008: 455-468.

[11]. Hvam L. A procedure for building product models[J]. Robotics and Computer-Integrated Manufacturing, 1999, 15(1): 77-87.

[12]. Van Deursen A, Klint P. Domain-specific language design requires feature descriptions[J]. Journal of Computing and Information Technology, 2002, 10(1): 1-17.

[13]. Natis R W S Y V. Service Oriented Architecture[J]. Gartner Group, SSA Research Note SPA-401-068, 1996, 2.

[14]. Luo M, Goldshlager B, Zhang L J. Designing and implementing Enterprise Service Bus (ESB) and SOA solutions[C]//Services Computing, 2005 IEEE International Conference on. IEEE, 2005, 2: xiv vol. 2.

[15]. Pahl C, Jamshidi P. Microservices: A Systematic Mapping Study[C]//CLOSER (1). 2016: 137-146.

[16]. Montesi F, Weber J. Circuit breakers, discovery, and API gateways in microservices[J]. arXiv preprint arXiv:1609.05830, 2016.

[17]. Kurhinen H, Lampi M. Micro-services based distributable workflow for digital archives[C]//Archiving Conference. Society for Imaging Science and Technology, 2014, 2014(1): 47-51.

[18]. Abbott M L, Fisher M T. The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise[M]. Pearson Education, 2009.

[19]. Bass L, Weber I, Zhu L. DevOps: A software architect's perspective[M]. Addison-Wesley Professional, 2015.