2020 2nd International Conference on Information Technology and Computer Application (ITCA) | 978-1-6654-0378-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ITCA52113.2020.00167

Enterprise service application architecture based on Domain Driven Model Design

Yu Ding
Solution Center
Meta-Sophia Research Institute
Beijing , China
660574@gdpr.com

LiLing Wang
Computer Application Technology Department
Research Institute of Petroleum Exploration & Development
Beijing , China
wll2018@petrochina.com.cn

Shaokun Li
Solution Center
Meta-Sophia Research Institute
Beijing , China

XuTong Wang
Solution Center
Meta-Sophia Research Institute
Beijing , China

JianWei Zhang
Solution Center
Meta-Sophia Research Institute
Beijing , China

***Abstract*—At present, each industry has invested huge energy in the development of its own information construction, and has also derived numerous high-quality enterprise information forms. At the same time, a variety of new information construction concepts are emerging. Among them, in order to solve the problems of the rapid development of information technology, such as difficult to express the business needs of enterprises, difficult to use the feature model, difficult to integrate large-scale systems. A concept of Domain Driven Design as web software design idea began to appear, which has formed a relatively unified design specification in the software development industry under the continuous improvement. Compared with the previous database driven development method, it emphasizes the concept of the domain, the architecture and the objects in it are clear, and the reusability is good. It breaks the idea of technology as the leader in traditional software design, and the complexity of software implementation lies in the domain itself rather than the specific implementation technology. In order to solve the integration platform project that a large-scale energy enterprise focuses on, this paper proposes a design scheme of Web Platform Based on Domain Driven Design. Through the analysis and design of the domain model in the case, using layered architecture and. Net Entity Framework, this paper elaborates the architecture design based on the domain model in the platform and the application implementation method using the architecture, and finally provides a general software development architecture mode for the design of large-scale integrated platform.**

Keywords-component; Domain-Driven Design; Architecture design; Micro-service Application; Model Design

I. INTRODUCTION

In the era of high integration of information and modernization, the cross integration of information in different fields of all walks of life has promoted great changes in the Internet software development industry. From the previous development models of process oriented, object-oriented and service-oriented architecture, it has gradually transformed into a new development process represented by rapid iteration, agile development and micro service. Because the success of products is more and more related to business complexity, external dependence, customer experience and other factors, the guidance of product realization has gradually changed from former developers to product designers, and the rapid technological change has led to various types of technology selection, which also puzzles the platform construction team. Therefore, the core goal of the whole modeling is how to make the design of the information platform have clear and rich descriptions of the software functional requirements, to truly achieve and meet the real intention of users, and to form a unified communication language and development specification in the development team.

Taking the domain model of an online bookstore as an example, a design scheme of building web platform based on Domain Driven Design is proposed [1]. It adopts the application advantages of MVC and Java EE to improve the re-usability, expansible and maintainability of MIS application development [2]. Based on the spring framework, this method implements a DDD plug-in based on domain message driven and memory modeling, which makes the project implementation fully compatible with DDD design [3]. It proposes a DDL method based on DSL to fully meet the requirements of the original and basic structure and behavior modeling of the actual software, and extends a basic action of expressing constraints and domain classes with annotations [4]. It can be seen that the above researches are related to design analysis model, Domain Driven Design and architecture implementation, but they are all feature domain model applications applicable to independent scene parts, and there is no specific scheme for the implementation of large-scale integration platform. Based on the existing research, this paper further studies the relevance of business flexibility analysis, enterprise level strategic model and large-scale underlying design, and then combines with the actual cases, in order to provide a general software development architecture mode for the design of large-scale integrated platform.

II. Advantages of Domain Driven Model Design Architecture

The domain model enables developers to express the complex software functional requirements, and the software can meet the real needs of users. On this basis, in the process of cooperating with product requirement designers to participate in project design, the whole complex software development process is accelerated by creating domain model, and the whole project life cycle is shortened. At the same time, based on the model, it can unify the communication language between developers and demand designers, and cooperate with the agile development concept of DEVOPS [5], so as to help the project team establish an organized, well-defined and collaborative environment in the implementation of large-scale complex platform.

A. *Domain Driven layered architecture design*

In the traditional object-oriented development process, the business implementation content is often written directly into the code of application layer and data access layer, that is, the developer abstracts the business logic content with the development logic at the early stage of development and implements it into the code. For example, the database model is built directly according to the module requirements, and the business logic implementation code is directly embedded in each component of the user interface [6]. The reason for this is to solve the short-term development work in the simplest way. However, it ignores a series of problems caused by the scattered logic implementation code in every corner of the whole framework.

Then, in the domain driven model, aiming at the above problems, a layered design concept is proposed [7]. Based on this concept, developers can design code implementation methods that meet their business needs, unify development specifications and later operation and maintenance standards, and make the program itself readable and easy to understand. In order to create a domain hierarchical framework that can handle complex logic, the most important core is separation. It refers to the explanation of Eric Evans in domain driven design, and separation makes every part of the design get separate attention. At the same time, it is necessary to maintain complex interaction within the system which is show in Figure 1:

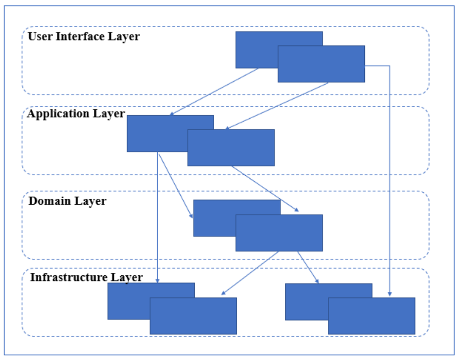


Figure 1. Standard Hierarchical Structure of Domain Model

User interface layer: complex external display of interaction information, interpretation of activity instructions, completion of data requests and other processing events, and links to the application layer and basic settings layer.

Application layer: this layer is used to define the tasks to be completed in the software, command and express the concept objects in the domain layer to solve the corresponding business logic events, and interact with other application layers for business data.

Domain layer: this layer is also the most core layer of domain driver, which requires developers to understand and transform business information and business rules into code and distinguish them.

Infrastructure layer: provide a complete data persistence mechanism for the above layers, and save the business state to the corresponding database.

When we study the layered architecture, we find that after implementing the previous code according to the above concepts, the design of each layer is clearer and more independent of each other. Developers can understand and modify the business rules well, and can easily capture the business knowledge content. In the process of implementation, domain design must be simplified as much as possible, so that the core intention of users can be clearly expressed after domain objects are combined.

B. *Strategic consistency of enterprise model construction*

Because this research is based on the large-scale integration platform of enterprise level, the construction of the model needs to achieve strategic level unified planning. But at the same time, the development of large-scale system is a very difficult and waste of resources in accordance with the absolute level of unity, which needs to consume a lot of manpower and time costs [8][9]. Therefore, it is necessary to carefully select which modules of the system can be separated, which modules should sink to the bottom of the platform, and which parts are strongly related to each other. According to the above problems, the project combines the suggestions of domain design driving to the large platform architecture, and takes the context relevance from business to code as the core point to carry out the overall architecture design.

Boundary Context: This model defines the standard of boundary context to better distinguish the application boundary range of each model. The scope of the model is a more limited part of the software system, which can only be used in one model, and try to maintain its unity. In the early stage of design, team members cut the whole business model to ensure that each model is a relatively complete object that can represent the business object description, minimize the gray area between codes, and then improve the reconstruction of objects in various fields.

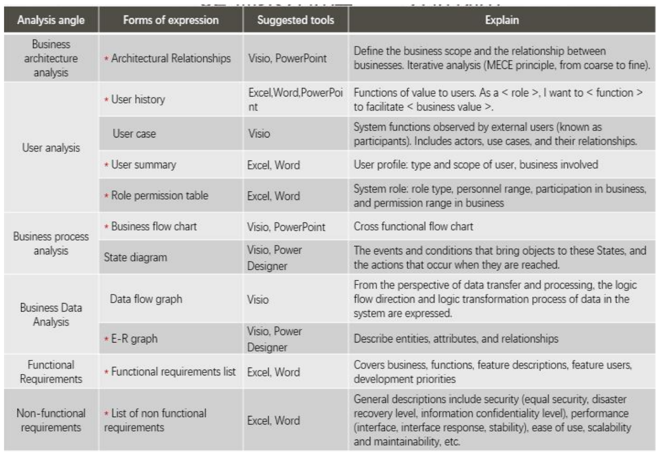
Context Map: In the development process, some developers are not very clear about the context of the boundary situation, which leads them to unconsciously make some scope spread operations, thus breaking the initial boundary conditions. If not controlled, these code modules can also overlap each other, and the boundary becomes more and more fuzzy, which is also a very dangerous situation. Then the model should focus on specific scenario applications, identify each model object that plays a role in the project, define its context description, naming rules, and clarify the content and management connection points that need to be shared in all messages. All of these require domain design experts, developers and business people to use a unified model language to describe the parts of business logic that need interaction, eliminate the gap in development, and ensure that domain model design can carry out correlation mapping as the general knowledge map.

Core Sharing： In the process of large-scale system construction, there are many methods and objects that need to sink to the bottom of the architecture for unified design, and completely separate from the business to ensure its high availability, high mobility and high re-usability [10]. When there are different teams in the same large-scale system, the sharing of core modules becomes particularly important, in order to ensure that the developed products can be effectively combined in the end. At the time of design, we plan and build a unified subset that is agreed to be shared by teams in the domain model, and share it as the core module. At the same time, the code subsets and database design subsets are also included in the special planning management. Any single modification to them is regarded as invalid, and unauthorized change is prohibited, which ensures the reliability of this part to the maximum extent.

III. FLEXIBLE DESIGN IN DOMAIN MODEL

As the end-user-oriented deliverable, the first acceptance standard of software is to meet all the requirements of users, and then to have the ability to re-factor and combine with each other [11]. Once the developers cannot fully understand the full meaning of the design scheme (including explicit and implicit rules), there will be project deviation, and it is difficult to test and track. In the face of this high-risk problem, designers need to have a clear planning scheme for the software as much as possible, to help developers to understand the business and code writing analysis, to reduce structural confusion. In order to make developers easy to understand and happy to use, and at the same time to adjust according to the change of requirements in time, we bring out a flexible design idea for designers and developers.

In domain driven design, we hope to see meaningful logical domains, so that the code can get the results we want after explaining the documents according to the execution rules, and at the same time, according to different analysis perspectives and forms of expression, we can face designers, developers and users in a graphical and visual way. The flexible analysis plan we designed is shown in Figure 2:



Analysis angle	Forms of expression	Suggested tools	Explain
Business architecture analysis	* Architectural Relationships	Visio, PowerPoint	Define the business scope and the relationship between businesses. Iterative analysis (MECE principle, from coarse to fine).
User analysis	* User history	Excel,Word,PowerPoint	Functions of value to users. As a < role >, I want to < function > to facilitate < business value >.
	User case	Visio	System functions observed by external users (known as participants). Includes actors, use cases, and their relationships.
	* User summary	Excel, Word	User profile: type and scope of user, business involved
	* Role permission table	Excel, Word	System role: role type, personnel range, participation in business, and permission range in business
Business process analysis	* Business flow chart	Visio, PowerPoint	Cross functional flow chart
	State diagram	Visio, Power Designer	The events and conditions that bring objects to these States, and the actions that occur when they are reached.
Business Data Analysis	Data flow graph	Visio	From the perspective of data transfer and processing, the logic flow direction and logic transformation process of data in the system are expressed.
	* E-R graph	Visio, Power Designer	Describe entities, attributes, and relationships
Functional Requirements	* Functional requirements list	Excel, Word	Covers business, functions, feature descriptions, feature users, development priorities
Non-functional requirements	* List of non functional requirements	Excel, Word	General descriptions include security (equal security, disaster recovery level, information confidentiality level), performance (interface, interface response, stability), ease of use, scalability and maintainability, etc.

Figure 2. Flexible analysis structure

Business analysis is to reorganize complex business requirements on the basis of business requirements refinement and classification. This flexible design delineates the business scope and defines the relationship between businesses. The design of business architecture can be carried out literately from coarse to fine, including business architecture analysis, user analysis, business process analysis, business data analysis, functional requirements and non-functional requirements [12]. With the help of the corresponding office tools for the transformation of logic to entity, after the split design, the developers can well understand and locate their coding purpose and direction, so as to clearly show the design intent of the model, so that users can easily see the running effect of the code. Therefore, it is easy to predict the result of code modification and minimize the side effects caused by dependency and confusion. At the same time, we design the most important parts of users in a fine-grained way, and maintain high design flexibility and coarse-grained planning for those parts that are not clear at the initial stage and may often need to change.

IV. IMPLEMENTATION OF ENTERPRISE PLATFORM BASED ON DOMAIN DRIVEN DESIGN

As we all know, it takes a long time for a software to go from idea to finished product to appearance. During this period, there will be risks from people, things, things and other aspects [13]. These risks make the complexity of software inevitable. Although there are many reasons for the complexity of software, it is impossible to avoid the complexity in the implementation process, but it can effectively control the complexity.

A finished software product needs the cooperation of product, development, testing, management, operation and maintenance and other roles to be successful. In the traditional working mode, each role is accustomed to thinking and analyzing software from the perspective of their own roles, which leads to ambiguity when each role looks at the same thing. In order to effectively control the complexity of software, and make all participants stand in the same angle, we introduce the concept of domain model. Domain model provides such a language that all participants can communicate and analyze without ambiguity, what they see is what they get.

In traditional software design, software is usually divided into logic layer and application layer [14]. In the domain driven design, first of all, we redefine the analysis perspective of developers, which requires that what you see is what you get in terms of business requirements. That is to say, when defining domain model, developers, demanders, testers, etc. should be clear at a glance. Seeing this model means understanding software. Secondly, using the mode of Code First, the warehouse layer is separated from the logic layer, which abstracts the interface of data persistence mechanism, and the logic layer is responsible for business logic control, which greatly reduces the amount of duplicate code. Finally, the mediator is introduced to decouple the code of application layer and logic layer, and the Restful [15] style is introduced to standardize the interface definition.

The implementation structure of the domain driven design scheme is shown in Figure 3. From the bottom up, infrastructure is the storage layer, which provides data persistence mechanism. Domain is the definition of domain object layer, which needs developers to analyze from the perspective of business. Application is the logic layer, which implements the business requirements logic. API is the application layer, which is responsible for interaction with external activities. Each layer is explained in detail below, and a real case (user's organization personnel management model) is used to show.

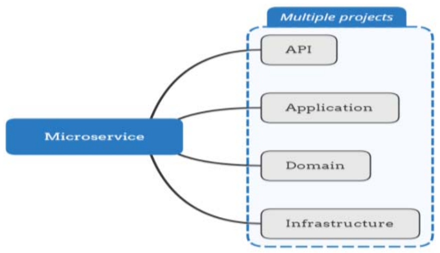


Figure 3. Example of Domain Driven Design

A. Domain Layer

Using domain driven design software, the first step is to consider the domain model, and to establish the domain model, we must first abstract the domain object. In the traditional way [16], developers are used to define from the abstract data structure, while the domain driven way requires developers to transform the analysis perspective from the business perspective, so that what they see is business.

This layer is explained below in conjunction with Figure 4 of the layer structure. The domain layer includes domain models and domain events. For code clarity, place the domain model under Aggregates/<X>Aggregates and the domain events under DomainEvents. It is recommended that the events be named <A>DomianEvent. A domain model is a model of a specific business and forms a domain object, which includes a data model, domain logic, or behavior, and a warehousing interface. For domain models, it is recommended to name the data model <X>Model. For its behavior capabilities, it is suggested to name <X>Behavior as an extension method. For warehouse interfaces, only add-delete methods are included. For interface implementations, at Infrastructure level, queries are sliced according to CQRS mode, it is recommended to name I <X>Repository, and for Aggregates/<X>Aggregates. In addition, domain objects contain aggregated objects, value objects, enumerated objects, where the aggregation root is a special aggregation object and serves as the only entry point for external access to aggregation.

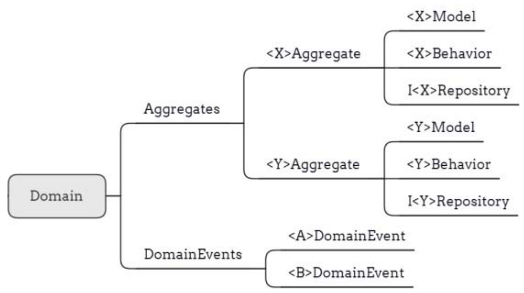


Figure 4. Domain layer diagram

For example, if a project wants to realize the management function of an organization model, it needs to analyze the organization objects. The design of its domain layer model can be shown in Figure 4: first, it has a detailed definition of specific name, unique code, superior organization, subordinate organization and other attributes, Secondly, it needs to have attributes that can support the interface to modify the name, delete the organization and other functions, and then for the unknown range that may appear in the future, it needs to be able to have extended reserved information location. In the traditional way, developers abstract the attribute and extended information into data model, and realize the function as the method of logic layer, so the model and database are basically the same structure. However, in the design scheme based on domain model, the business is regarded as the object of model analysis, so the attribute of mechanism is regarded as the data model of domain model, and the function is regarded as the behavior and interface of domain model. Therefore, it will be found that in this scenario, the domain model and the database model are usually inconsistent.

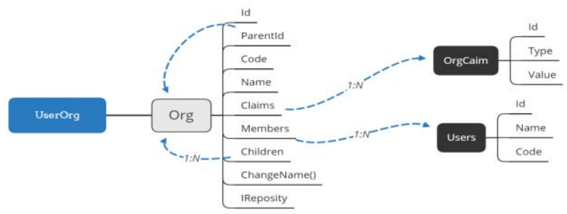


Figure 5. Example of domain level design

B. Infrastructure Layer

In general software design examples, the Infrastructure layer is almost mixed with the business logic. In this scheme, the two are divided and read-write separation is implemented according to Command Query Responsibility Segregation （CQRS） mode [17]. Since the operation of data "writing" is modeled in the domain layer, each layer can focus on its own domain to make the data structure clearer. The Infrastructure layer mainly includes: Infrastructure model (database model) configuration, Infrastructure access context, "write" interface implementation (the interface is defined in the domain layer). The layers are shown in Figure 6, and the files associated with the warehouse model are in the EntityFramework. The warehouse model configuration recommendation is named <X>ModelConfiguration, the warehouse access context recommendation is named <DB>Context, and all migrations generated using Code First mode are in Migrations. The storage interface integrated in the Domain layer is recommended to be named <X>Repository and placed in Repositories.

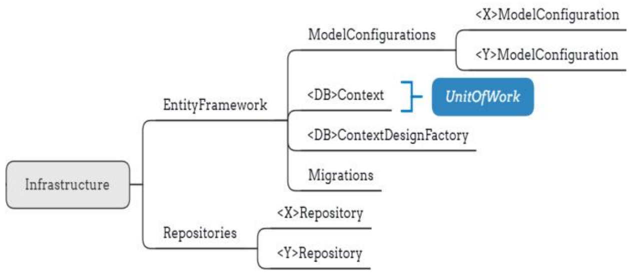


Figure 6. Infrastructure layer structure diagram

As an example, the domain model structure in Figure 4 defines the members and claims attributes. However, in its warehouse model configuration, these two attributes are not defined as fields of database tables, but as associated tables, as shown in Figure 5. For claims attribute, it represents the collection of extension information of the organization, such as province, level, sorting, etc., so we can define it with a foreign key association table. For members attribute, it represents the collection of all personnel information under the organization, which requires the association with another principal user. Therefore, the model defines a separate association table between organizations and personnel to represent.

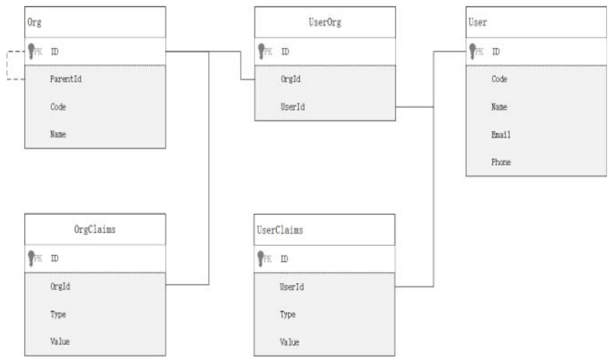


Figure 7. Structure example of Infrastructure layer

C. Application Layer

As mentioned earlier, this layer contains processing logic for all businesses. Most developers, after separating the reservoir from the layer, are accustomed to mixing the layer directly with the API (interface layer), but this can lead to more and more API layers and confusion in structure. In this scenario, the concept of an intermediary is introduced to decouple this layer from the API layer to form a structure of light API and heavy application. Intermediator mode [18][19]: A mediator encapsulates a series of object interactions, so that the coupling is loose and the interaction between them can be changed independently. The purpose is to decouple message sending and processing. It supports the use of synchronous or asynchronous modes to publish messages, create and listen for events in both unicast and multicast formats. Messages are thus defined as commands and events, which are one-to-one unicast relationships and events as one-to-many multicast.

1) Command Request - Command Processing: A one-to-one relationship

Define the Command class to pass API layer message data, all of which are single line model properties. Therefore, a format named <X><Action>Command is recommended. Command processing is the processing logic that implements all business, and it is recommended that you name it <X><Action>CommandHandler. For example, to create an organization, we need to define two classes, the OrgCreateCommand and the OrgCreateCommandHandler. In the first class, we define the necessary attributes, such as name, code, to create an organization. The second class contains specific business logic processing, such as processing data in the Command class to conform to the database structure, invoking the reservoir interface for insertion. In this scenario, we focus this on Commands, as shown in Figure 8.

2) Event Definition - Event Processing: One-to-Many Relationship

Event Definition Module refers to the processing logic that defines the Command class to pass API layer message data. Event processing module events occur with multiple business processes that need to be generated. Events are divided into domain events and integration events. Integrated events are defined and processed at the application level, and integrated event processing is triggered by calls outside the realm. Domain events are defined at the domain level, processed at the application level, and triggered by mediator extension calls. As shown in Figure 8. In this layer, the scenario places integrated events and processes in IntegrationEvents and IntegrationEventHandlers. Domain event handling is placed in DomainEventHandlers. In the case of realm events: modifying the name of an organization (ChangeName() method defined at the Domain layer), we define an event class, OrgNameChangedEvent. It is divided into two event handling subclasses: OrgNameChangedEventHandler1 and OrgNameChangedEventHandler2, and then defines the necessary properties of the event in the event class: the new name, the institution ID. Update agency data tables in event processing class 1 and modify user agency associated tables in event processing class 2. Queries as shown in Figure 8.

The operation of the new agency data table, which modifies the user agency association table in Event Processing Class 2. In addition, the previous section indicates that the interface definition of the Domain layer to the domain model only contains a method of "write" operation, and the query is sliced according to the CQRS mode, so this layer also includes the query interface to aggregated objects and its implementation.

3) Query Model - Query Interface - Query Interface Implementation

Query model: Query model and command and event request model are independent of each other. Naming <X>QM is recommended. Query interface: It is recommended to name I<X>Queries. Query interface implementation: It is recommended that you name <X>Queries, note that instead of using a warehouse interface, you use a DbContext derived class directly. To summarize, the model design of this layer is fully demonstrated in Figure 8.

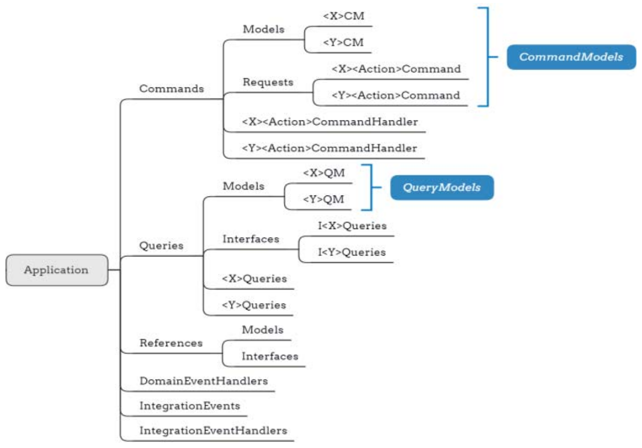


Figure 8. Application Layer Structure Diagram

D. *Interface layer*

This layer includes the definition of the external interface. After introducing the mediator mode, the interface request class only relies on the mediator, changing the original one-to-many dependency into one-to-one dependency, reducing the dependency between classes and, of course, reducing the coupling between classes. This layer can then connect with the previous three layers to complete user requests simply by sending messages through the intermediary. This layer structure is shown in Figure 9. Controllers contain all controllers and define all external interaction interfaces. Models contain all view models and are recommended to be named <X>VM.

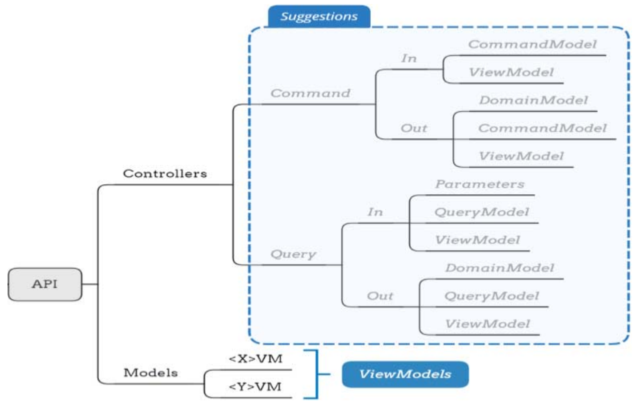


Figure 9. API layer structure diagram

The Restful style [20] is also introduced in this layer to surround the interface with resource definitions, following the principles:

a) Defined URLs should focus on business resources, and resource URLs should be noun-based, not verbs.

b) If an operation cannot be mapped to a specific resource, it can be done by adding Query Options.

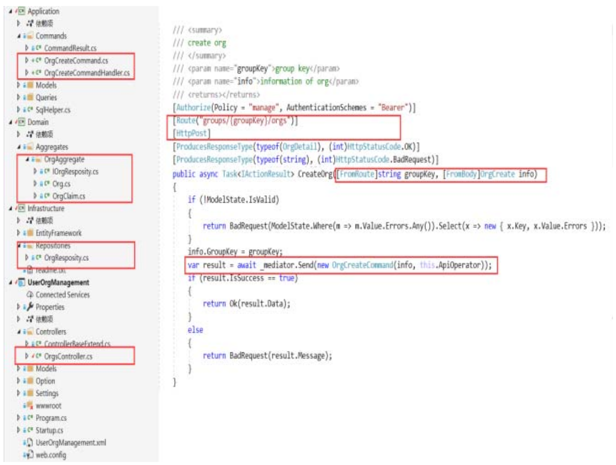
c) Use HTTP Method to specify specific operations.

d) Similarly, naming suggestions and priority principles for controller action method parameter types are as follows:

e) CommandModel takes precedence over ViewModel for command execution as an input parameter in an action method. The output takes precedence over DomainModel, followed by CommandModel, and finally ViewModel.

f) For query execution, the input parameters in the action method take precedence over the query parameters, followed by QueryModel, and finally the view model ViewModel. Output takes precedence over DomainModel, followed by QueryModel, and finally ViewModel.

For example, to create an organization, first define the model Org class in the Domain layer, and then implement the interface to insert the database in the Infrastructure layer. Then, in the Application layer, define the OrgCreateCommand class that creates the organizational command model and the OrgCreateCommandHandler class that handles commands. Finally, the interface is defined at the API level, and the interface request type is the OrgCreate class. Once the code structure is established in the above way, the request path for the specific interface is http://xxx.com.cn/groups/{groupKey}/orgs, the request mode is HttpPost, the request parameter is groupKey, the request body parameter info (request type is OrgCreate class), and the command model (OrgCreateCommand) is declared in the controller method. So that the mediator is used to send messages and get and return results. The schema structure and interface definitions are shown in Figure 10.



```
/// <summary>
/// create org
/// </summary>
/// <param name="groupKey">group key</param>
/// <param name="info">information of org</param>
/// <returns></returns>
[Authorize(Policy = "manage", AuthenticationSchemes = "Bearer")]
[Route("groups/{groupKey}/orgs")]
[HttpPost]
[ProducesResponseType(typeof(OrgDetail), (int)HttpStatusCode.OK)]
[ProducesResponseType(typeof(string), (int)HttpStatusCode.BadRequest)]
public async Task<IActionResult> CreateOrg([FromRoute]string groupKey, [FromBody]OrgCreate info)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState.Where(m => m.Value.Errors.Any()).Select(x => new { x.Key, x.Value.Errors }));
    }
    info.GroupKey = groupKey;
    var result = await _mediator.Send(new OrgCreateCommand(info, this.ApiOperator));
    if (result.IsSuccess == true)
    {
        return Ok(result.Data);
    }
    else
    {
        return BadRequest(result.Message);
    }
}
```

Figure 10. Structure of interface layer and interface

V. CONSTRUCTION OF INTEGRATED PLATFORM FOR MULTI-BUSINESS SYSTEMS

Compared with the traditional construction method of single business system, multi-system business integration project construction more tests the project team's overall control ability. At present, after practical application, a set of platforms for data interaction and unified integrated access to multiple business systems has been successfully built through domain-driven model. Solves the problems of large time span, inconsistent technical architecture, low system scalability, low flexibility, high maintenance complexity and high cost of operation and maintenance. It avoids the duplicate development of similar functions in each system, has a low reusability rate of IT assets, and is not conducive to saving the investment cost of information technology.

The overall solution includes support and compatibility for a variety of front-end and back-end frameworks, can use a variety of middleware to respond to office management business needs, can use a variety of languages for service development, and the development process follows mainstream standard protocols. Front-end applications support multiple frameworks such as Vue, Nodejs, Jquery, RequireJS, Bootstrap, and back-end services support frameworks such as Net Core, Java. Websphere, RabbitMQ, Redis, IIS, Nginx and other middleware can be used flexibly. Development languages support applications developed in Java, Python, C# and many other languages, depending on the specific situation of different development teams and business systems. Standard protocols include Http, Https, Json, Rest, Openxml and many others.

The project technical plan takes full account of the support and utilization of open source, and takes full advantage of the compatibility of domain-driven models with third-party open source software, such as Redis, RabbitMQ, Vue, Nginx, etc., to save project costs while ensuring the quality of system construction and subsequent operations and maintenance services. At the same time, the IPaaS layer basic management software solution is mature, using stable commercial software and services, to ensure the stable operation of the integration platform and related business systems.

VI. Conclusion

In this study, we have learned from the previous construction experience from the analysis model, which has helped us to understand the domain-driven model concept. For domain-driven design of large enterprise integration platform, the core part is always around how to solve complex business design problems, how to grasp business logic and convert it into business logic model, and how to solve the reusability and reconfiguration ability of the system in the future continuous construction process. From a practical point of view, the rules defined by domain drivers are clear and easy to understand, especially for graphic logic design. It handles business information better than previous design frameworks and is more flexible in concept, so its rules can be widely applied across platforms. In addition, in order to further plan domain objects, some special contextual methods have been applied to projects, such as boundary definition, mapping, and core sharing. Therefore, the research results of domain-driven design play an important role in the further maturity of modern software development. This paper also provides a generic software development architecture mode for the design of large integrated platforms.

References

[1] Duc Minh Le, Duc-Hanh Dang, Viet-Ha Nguyen. (2019) Generative software module development for domain-driven design with annotation-based domain specific language. Information and Software Technology. Vol.120.

[2] Bondarenko Vitaliy Ivanovich, Bilousov Vyacheslav Vladimirovich, Nedopekin Fedor Victorovich,Bodriaha Viktor Viktorovich, Antropova Larisa Vitalievna. (2019) Using MVC pattern in the software development to simulate production of high cylindrical steel ingots. Journal of Crystal Growth. Vol.526.

[3] Florian Rademacher, Jonas Sorgalla, Sabine Sachweh. (2018) Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. IEEE Software. Vol.35 (3), pp.36-43

[4] Duc Minh Le, Duc-Hanh Dang, Viet-Ha Nguyen. (2018) On domain driven design using annotation-based domain specific language. Computer Languages, Systems & Structures. Vol.54.

[5] DENG Jingying, HUANG Sui . (2004) Agile Development:Practices for eXtreme Programming in Developing Management Information System. Computer Engineering. 30(24):189-191.

[6] Einar Landre , Harald Wesenberg , Jorn Olmheim. (2007) Agile enterprise software development using domain-driven design and test first.Object-oriented programming systems and applications companion. 12(4):983–993

[7] Dmitry Petrashko, Vlad Ureche, Ondřej Lhoták. (2016) Architectural improvement by use of strategic level domain-driven design. Object-Oriented Programming, Systems, Languages, and Applications. 36(4):809–814.

[8] ZHU Ai-hong, YU Dong-mei, ZHANG Ju-li. (2005) Research on B/S-based software architecture. Computer Engineering and Design. 26(5):1164-1165.

[9] Jin Lingzi. (1998) PROGRESS IN TESTING OBJECT ORIENTED SOFTWARE. Journal of Computer Research and Development. 35(1):6-13.

[10] JIN Ling-Zi, ZHU Hong. (1998) A USE SCENARIO DRIVEN APPROACH TO AUTOMATIC GENERATION OF SOFTWARE REQUIREMENTS MODELS. Chinese Journal of Computers. 21(8):673-681.

[11] Wang Peijin. (2001) Theory of Object－Oriented Free Design and Free Control（I）. Computer Engineering and Applications. 37(16):17-19.

[12] Yang Jie. (2019) Application Architecture Design Based on Top-Down Design Thoughts. Modern Information Technology. 3(20):134-135+138.

[13] Jhon Masso, Francisco J. Pino, César Pardo,Félix García, Mario Piattini. (2020) Risk management in the software life cycle: A systematic literature review. Computer Standards & Interfaces. Vol.120.

[14] Wang Zhigang. (2019) Application of Software Engineering Technology in System Software Development. China Computer & Communication. 31(24):41-43.

[15] Paczian Tobias, Trimble William L, Gerlach Wolfgang, Harrison Travis, Wilke Andreas, Meyer Folker. (2019) The MG-RAST API explorer: an on-ramp for RESTful query composition. BMC bioinformatics. 20(1).

[16] Li Bing. (2018) Research on Program Analysis and Verification of Complex Data Structure Programs. Nanjing University.

[17] Feng Peipei. (2016) Research on and Implementation of CQRS/ES Based on REST. Journal of Heihe University. 7(7):219-220.

[18] Luo Da, Zhang Yuanzhou, Zhou Xiaocong. (2008) Implementation of Mediator Pattern Based on AOP. Computer Applications and Software. (11):46-47+111.

[19] Hu Yu. (2008) A Method of Database Real-time Synchronization Based on Mediator Pattern. JiLin University.

[20] Long Jun. (2018) Research on distributed application development based on restful Web Service. Computer Knowledge and Technology. 14(35):87-89.