PinFaq

Login

Ask a Question

Domain Driven Design (DDD) architecture layer design for CRUD operation

+1
vote

asked May 14, 2014 in Software Practices by Raja Chandrasekaran

Now a days I am hearing a lot about domain driven design (DDD). I am in the process of learning the domain modeling. Even though I have some idea about the domain modelling part, I could not really find a good architecture design which explains me, what are the layers should be there and what is the purpose, role and responsibility of those layers. Could someone post any diagram or explanation to understand the complete DDD architecture design. It would be better if the design is without using any ORM like Entity Framework or N-Hibernate. Thanks in advance

ddd csharp design-pattern software-practices

Share

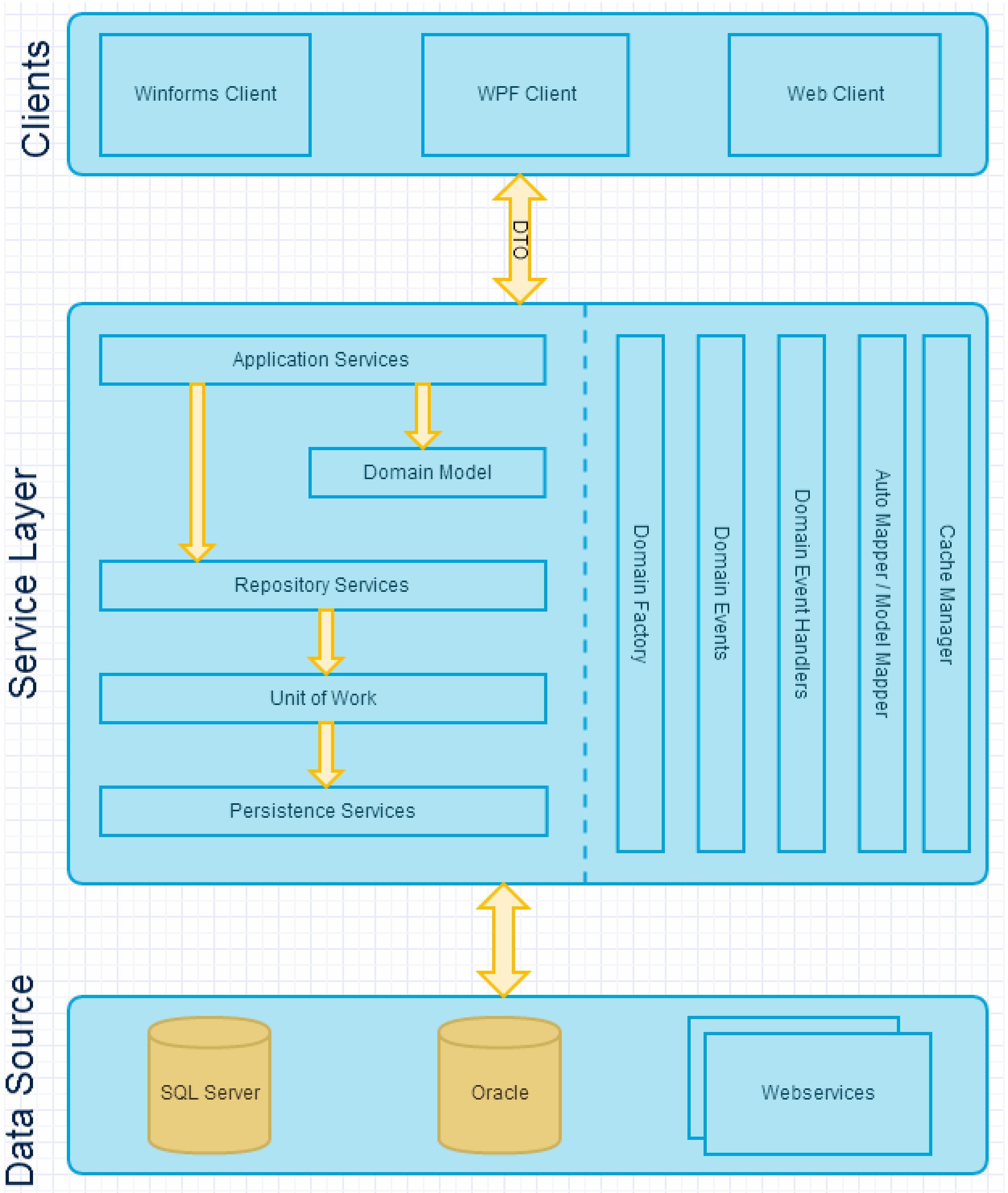
answer

1 Answer

+1
vote

✓

answered May 24, 2014 by Raja Chandrasekaran
selected May 24, 2014 by administrator


Clients
Winforms Client
WPF Client
Web Client
DTO
Service Layer
Application Services
Domain Model
Repository Services
Unit of Work
Persistence Services
Domain Factory
Domain Events
Domain Event Handlers
Auto Mapper / Model Mapper
Cache Manager
Data Source
SQL Server
Oracle
Webservices

Clients: Client application could be really any technology if the service layer is exposed as web-service / wcf-service. So, the client would be very thin layer which is responsible for

1. Read user input and send it to application services to perform CRUD operation.
2. Get information from application service and display those in user interface.

Application Service: This will be acting as façade and accepts any request from clients. Once the request comes, based on the operation, it may call Domain Factory to create domain model, or calls repository service to re-create the existing model. Conversion between, DTO (Data Transfer Object) to domain model and Domain Model to DTO will be happening here. Application service can also call another application service to perform additional operations.

Domain Model: This is the core of DDD (Domain Driven Design) and this is the place where all your business logics, business validation and business behaviors will be implemented, in fact this is the only place you should implement business logics. Objects and behaviors are tightly coupled in DDD and the Aggregate Roots, Entities and Value Objects will help to achieve this.

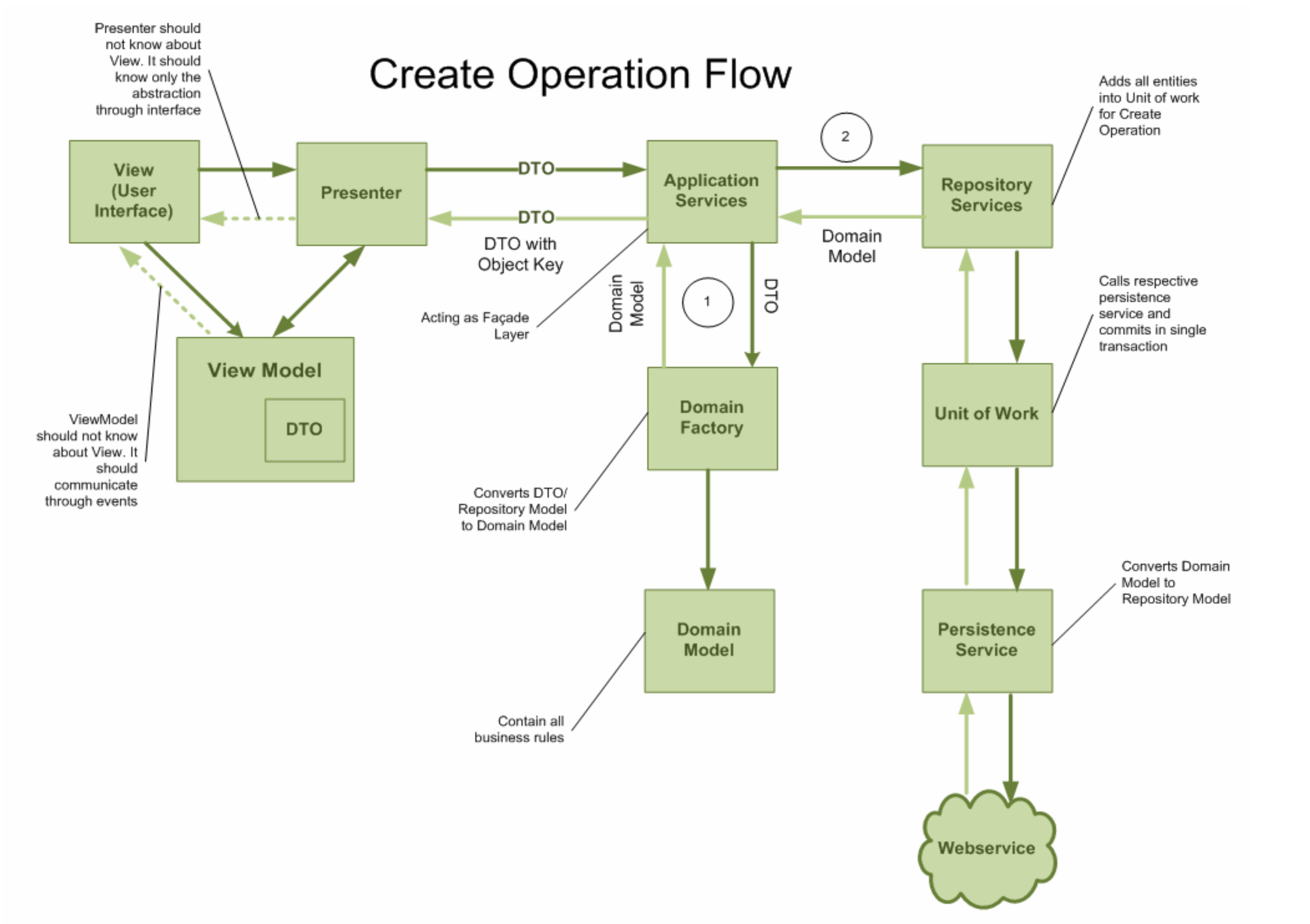
Domain Factory: This layer will be responsible for creating / recreating domain objects from DTO / Database response / web service response. This layer should be used only to create domain objects. We should write any update domain functionalities here. If you look at the flow sequence diagram given below you will understand.

Repository Service: Repository services are used to store the domain model data into data store. But it will identify whether the domain object is a new object or modified object or the domain object has to removed etc. Once the repository services identify the object status, it will add those objects to Unit of Work. Another import thing is, the Repository services must always work only with the Aggregate Root. Because in DDD, entities or value objects which are inside an aggregate root cannot be accessed directly.

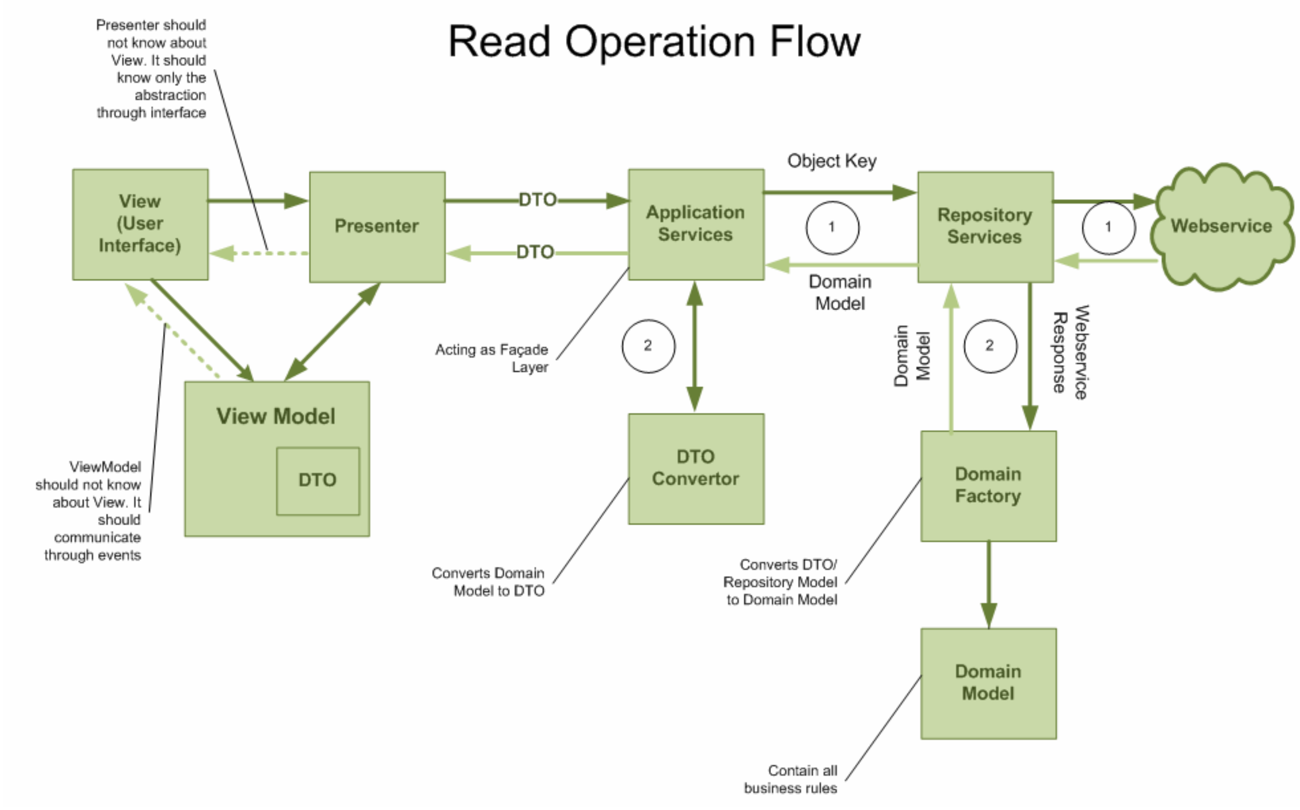
Unit of Work: This will be collecting the objects from repository services and calls respective Persistence service to INSERT, UPDATE or DELETE that object. Responsibility of Unit of Work is to process all request in a single transaction and make sure that, if the process is success or failure keep the object in a valid state (No intermediate state). i.e If any one persistence service fails, the whole transaction will be failed so, roll back operation will be called to put the object back in initial state.

Persistence Service: This is the layer where persistence operations (INSERT, UPDATE and DELETE) will be implemented. This layer may work with SQL Server / Oracle / Web service / WCF service to persist the data. READ operation can be implemented in the Repository Services itself.

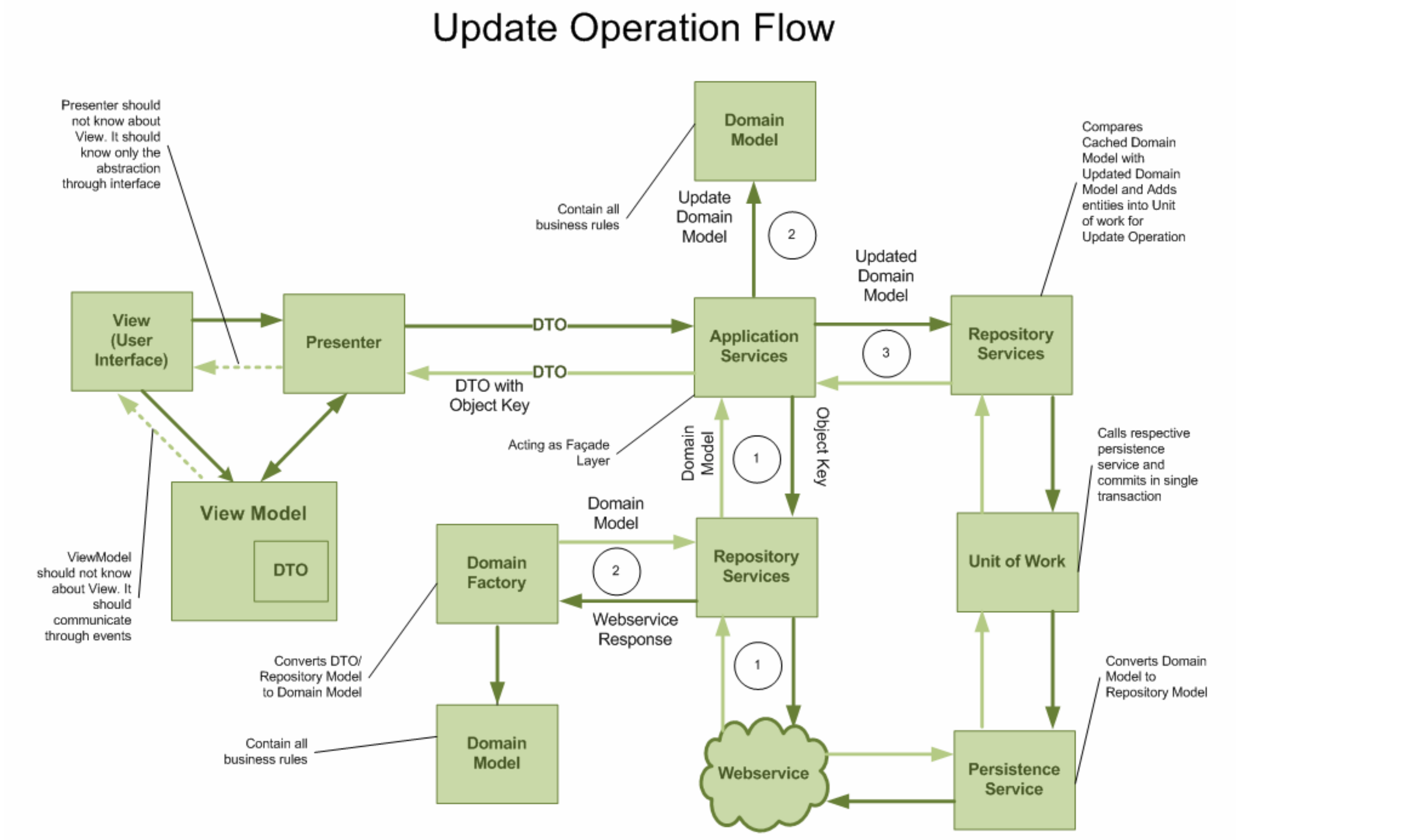
CRUD Operations Flow


Create Operation Flow
Presenter should not know about View. It should know only the abstraction through interface
View (User Interface)
Presenter
DTO
DTO
DTO with Object Key
Application Services
2
Repository Services
Adds all entities into Unit of work for Create Operation
Domain Model
Acting as Façade Layer
Domain Model
1
DTO
View Model
DTO
ViewModel should not know about View. It should communicate through events
Calls respective persistence service and commits in single transaction
Unit of Work
Domain Factory
Converts DTO/ Repository Model to Domain Model
Domain Model
Contain all business rules
Persistence Service
Converts Domain Model to Repository Model
Webservice

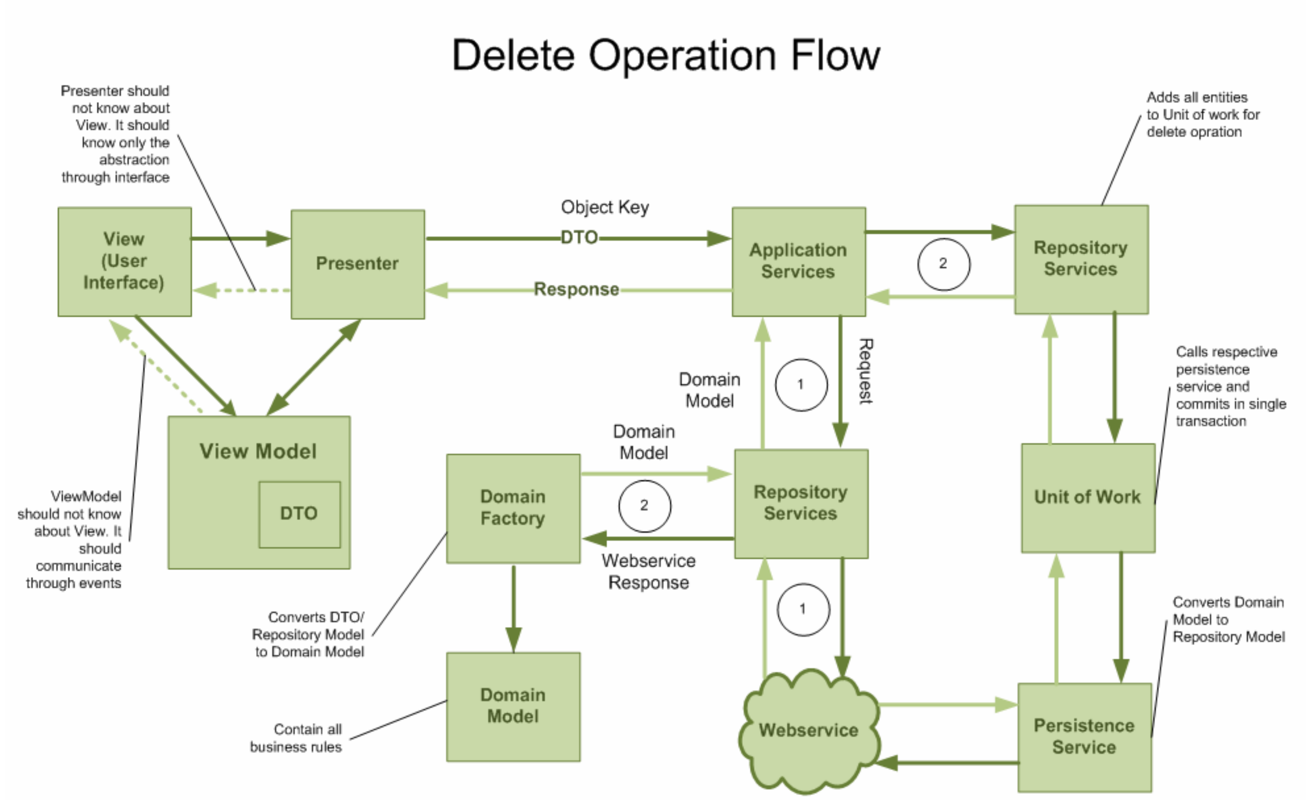
Create Operation Flow: In the current design, I have used MVP-VP design pattern which is nothing to do with Domain Driven Design. First the call will be requested from client. Presenter will prepare the DTO object and send it to Application Service. Application service will call Domain Factory to create the Domain Model. Once the model is created, that mean all of your business rules are satisfied (If any one of the rules is not satisfied, then the domain model object should throw exception. Object should not be created). Domain Factory will send the Domain Object to application service, and the application service will call Repository Services and send the Domain Object. Repository service will identify the status of the object and send it to Unit of work along with the Persistence Service which should be used. Once it is done, then the Application Service will call the SaveChanges() method in Unit of work. Now unit of work will execute operation for each object added by the Repository Services and calls the respective persistence service to execute the operation.


Read Operation Flow
Presenter should not know about View. It should know only the abstraction through interface
View (User Interface)
Presenter
DTO
DTO
Application Services
Object Key
1
Domain Model
Repository Services
1
Webservice
View Model
DTO
ViewModel should not know about View. It should communicate through events
Acting as Façade Layer
2
DTO Convertor
Converts Domain Model to DTO
Domain Model
2
Webservice Response
Domain Factory
Converts DTO/ Repository Model to Domain Model
Domain Model
Contain all business rules

Read Operation Flow: Presenter will send the key of aggregate root which should be retrieved to Application Service. Application service calls the repository service to fetch data from database. Once the response come from database, Repository service will call Domain Factory to create the Domain Model from database response. Once the Domain Factory created the Domain Object and returned back to repository service it will send it to application service. Since all the communication between Application Service and the Presenter of the client application is using DTO, application service will call Model Mapper to convert the Domain Object to DTO and send it back to presenter.


Update Operation Flow
Presenter should not know about View. It should know only the abstraction through interface
Domain Model
Contain all business rules
Update Domain Model
2
Updated Domain Model
Compares Cached Domain Model with Updated Domain Model and Adds entities into Unit of work for Update Operation
View (User Interface)
Presenter
DTO
DTO
DTO with Object Key
Application Services
3
Repository Services
Acting as Façade Layer
Domain Model
1
Object Key
Calls respective persistence service and commits in single transaction
View Model
DTO
ViewModel should not know about View. It should communicate through events
Domain Model
Domain Factory
2
Repository Services
Unit of Work
Webservice Response
1
Converts DTO/ Repository Model to Domain Model
Converts Domain Model to Repository Model
Contain all business rules
Domain Model
Webservice
Persistence Service

Update Operation Flow: Presenter will prepare the DTO object and send it to Application Service. Application service will get the Key of the aggregate root from the Data Transfer Object and calls the repository service to fetch data from database. Once the response come from database, Repository service will call Domain Factory to create the Domain Model from database response. Once the Domain Factory created the Domain Object and returned back to repository service it will send it to application service. Not application service will directly work on the domain object and update the domain object with the data in the DTO that came from presenter. Once the Domain Object updated, the application service will call Repository Services and send the Domain Object. Repository service will identify the status of the object and send it to Unit of work along with the Persistence Service which should be used. Once it is done, then the Application Service will call the SaveChanges() method in Unit of work. Now unit of work will execute operation for each object added by the Repository Services and calls the respective persistence service to execute the operation.


Delete Operation Flow
Presenter should not know about View. It should know only the abstraction through interface
View (User Interface)
Presenter
Object Key
DTO
Response
Application Services
2
Repository Services
Adds all entities to Unit of work for delete opration
View Model
DTO
ViewModel should not know about View. It should communicate through events
Domain Model
1
Request
Domain Model
Domain Factory
2
Repository Services
Webservice Response
Unit of Work
Calls respective persistence service and commits in single transaction
Converts DTO/ Repository Model to Domain Model
1
Domain Model
Contain all business rules
Webservice
Persistence Service
Converts Domain Model to Repository Model

Delete Operation Flow: Presenter will send the key of aggregate root which should be Deleted to Application Service. Application service calls the repository service to fetch data from database. Repository service fetch data from db and calls Domain Factory to create the Domain Model. Domain Factory creats Domain Object and return it back to repository service. Then repository service adds that domain object into Unit of Work in "Delete" status. Once it is done, the Application Service will call the SaveChanges() method in Unit of work to execute delete operation to delete the entire aggregate including entire child elements.

comment

Your answer

Preview

Your name to display (optional):

✓ Email me at this address if my answer is selected or commented on:

Privacy: Your email address will only be used for sending these notifications.

Anti-spam verification:

To avoid this verification in future, please log in or register.

Add answer

Most popular tags

csharp c# wpf jquery php web-technology sql-server wcf xaml data-binding

Follow us

Related questions

How does MVVM-Model-View-ViewModel work in WPF

What is Abstract Factory Pattern? Where should I use?

Difference between Factory Method Pattern and Abstract Factory Pattern

What is factory pattern? where should I use?

Unit Of Work pattern with repository pattern and entity-framework

Repository Pattern using Entity Framework

What is MVPVM design pattern? How to implement in Winforms application What is SOLID Principle

Winforms exception Cross-thread operation not valid Best explanation for why to use delegate in dotnet applications

All categories

Information Technology

- Software Practices
- ASP.Net MVC
- C#.NET
- Winforms
- VB.NET
- WPF
- WCF
- Workflow Foundation
- Entity Framework
- SQL Server
- Php
- MySQL
- JQuery
- javascript library
- bootstrap
- Visual Studio
- Other

Send feedback | XML Sitemap Powered by Question2Answer | Snow Theme by Q2A Market