ТЕХНИЧЕСКИЕ НАУКИ

Костевич Алексей Андреевич
бакалавр Белорусского государственного университета
информатики и радиоэлектроники,
инженер-программист
ИООО «ЭПАМ Системз»
г. Минск, Республика Беларусь

АНАЛИЗ ПРОБЛЕМ ПРОЕКТИРОВАНИЯ УРОВНЯ ДОСТУПА К ДАННЫМ НА ОСНОВЕ ПРИНЦИПОВ DDD

Аннотация*: в статье кратко рассматриваются основы проектирования модели предметной области на основе концепций Domain Driven Design. Выявляются и анализируются проблемы проектирования уровня доступа к данным для объектов предметной области. Дается краткий анализ использования существующих технологий для реализации данного уровня. Как результат, представлено решение для построения масштабируемого и эффективного решения для хранения и доступа к данным данных в соответствии с правилами и концепциями DDD.*

Ключевые слова*: модель предметной области, предметно-ориентированное проектирование, уровень доступа к данным, реляционная база данных, NoSQL, агрегация.*

На сегодняшний день программное обеспечение разрабатывается и используется для автоматизирования все более широкого круга процессов реальной жизни. Естественная сложность моделируемых процессов и предметных областей в целом требовали поиска новых подходов к проектированию систем. Ответом на неуклонно возрастающую сложность бизнес правил и процессов стало появление концепции DDD (сокр. от англ. Domain Driven Design – «предметно-ориентированное проектирование»).

DDD, вместо существующего подхода проектирования модели под конкретную архитектуру, использует противоположный подход: «проектирование по модели», ставя на первой место модель предметной области.

Таким образом, уровни стандартной многоуровневой архитектуры рассматриваются в терминах DDD, как инфраструктура, необходимая для работы модели.

Организация хранения объектов предметной логики является одним из наиболее проблемным и обсуждаемых механизмов. Чтобы выявить, в чем заключается проблема проектирования, необходимо рассмотреть базовые концепции построения модели.

В соответствии с DDD, модель предметной области строится из трех основных элементов.

1. Объект-значение (Value Object) – элемент модели, полностью определяющийся своими атрибутами [1, с. 103]. Объекты-значения не имеют своей индивидуальности и описывают признаки. Примерами таких элементов могут быть цвет, высота, тег.

2. Сущность (Entity) – элемент модели, обладающие своей индивидуальностью и временем существования [1, с. 96]. Сущности всегда имеют идентификатор, позволяющий отличить их от других сущностей. Например, близнецы, несмотря на одинаковые значения своих атрибутов, являются разными сущностями: их уникальность определяется идентификатором.

3. Корень агрегации (Aggregate Root) – сущность, включающая в себя другие взаимосвязанные сущности и объекты-значения и представляющая собой неделимое целое с точки зрения изменения данных. Другими словами, корень агрегации – сущность, ответственная за соблюдение всех инвариантов своих внутренних элементов [1, с. 126].

Согласно правилам DDD, вся работа с моделью предметной области должна производиться только через корни агрегации, так как они несут ответственность за проверку инвариантов. Нигде за пределами агрегата не может храниться ссылка на что-либо внутри него. Из этого следует, что только агрегаты могут извлекаться из базы данных и сохраняться в ней, т.е. агрегат представляет собой границы транзакции [1, с. 127].

Стандартный подход многоуровневой архитектуры, применимый для построения CRUD (сокр. от англ. Create, Read, Update, Delete) систем кажется на первый взгляд, приемлемым решением для организации хранения агрегатов (рис. 1).

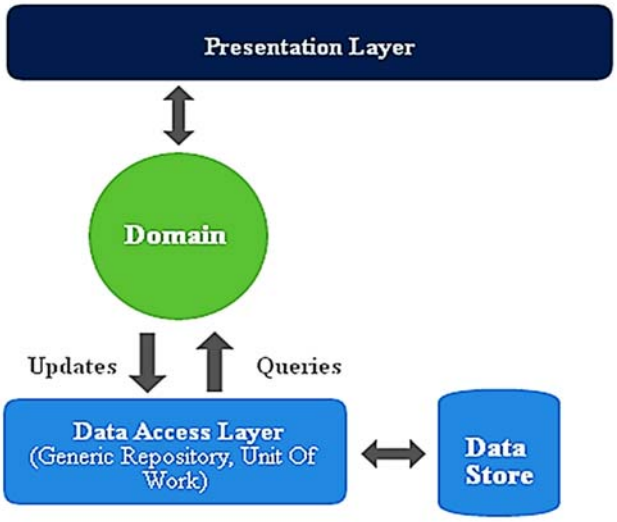

Рис. 1

Действительно, ведь реляционные базы данных вместе с ORM (сокр. от англ. Object-Relational Mapping – «объектно-реляционное отображение») или современные документно-ориентированные базы данных (MongoDB, RabenDB) позволяют организовать хранение агрегатов, а паттерны проектирования –спроектировать уровни абстракции и построить необходимую инфраструктуру. Однако анализ такой архитектуры показывает, что она является мало пригодной на практике и содержит концептуальные ошибки. Рассмотрим основные проблема данного подхода.

1. При проектировании уровня представления (Presentation Layer) в большинстве случаев требуется извлечение не агрегатов, а моделей представления (view models). Агрегаты предназначены для соблюдения инвариантов, а не для отображения данных. Извлечение агрегатов для построения уровня представления – крайне неэффективный и слабо масштабируемый подход. Из-за этого следует, что уровень доступа к данным должен предоставлять инфраструктуру как для записи агрегатов, так и для извлечения сложной модели представления.

2. Операции, применимые к модели предметной области, не симметричны относительно операций CRUD. Использование таких подходов как, например, Generic Repository, часто оказывается невозможным и является признаком анемии спроектированной модели. Проектируемая модель может не поддерживать удаления корня агрегации (Delete), но поддерживать его деактивацию (Deactivate) или архивирование (Archive), или не поддерживать ни одну из них вовсе. Таким образом, операции CRUD не являются реальными операциями предметной области и не могут в общем виде быть на них отображены.

3. DDD – намного более глубокая теория, чем может показаться на первый взгляд, и естественным путем решает множество существующих проблем CRUD систем. Так, паттерн Unit Of Work, повсеместно встречающийся в реализациях уровня доступа к данным для организации транзакций, является не только бесполезным для систем, построенных на основе DDD, но и нарушает принципы предметно-ориентированного проектирования [5]. Возможность обновления нескольких агрегатов в рамках одной транзакции для обеспечения целостности данных, предоставляемая паттерном Unit Of Work, просто не имеет смысла в теории DDD, так как агрегат – уже является естественной границей транзакции и соблюдения инвариантов.

Кроме вышеописанных проблем существует ряд ограничений существующих инструментов и технологий, значительно усложняющих процесс «проектирования по модели».

1. Реляционные базы данных не предназначены для хранения агрегатов. Данная проблема известна под названием «Object-Relational Impedance Mismatch». Одним из решений данной проблемы является технология ORM (сокр. от англ. Object-Relational Mapping – «объектно-реляционное отображение»). Однако существующие ORM библиотеки (Hibernate, Entity Framework) выдвигают определенные требования к модели предметной области и делают невозможным обеспечение ее инкапсуляции (а значит и соблюдение инвариантов) [6]. Это проблема может быть решена с помощью дополнительных уровней абстракции, однако данное решение не всегда целесообразно.

2. Документно-ориентированные базы данных позволяют работать непосредственно с агрегированными данными (документами), однако отсутствие полноценной поддержки принципов ACID (сокр. от англ. Atomicity, Consistency, Isolation, Durability) между коллекциями документов значительно затрудняет их применение. Несмотря на то, что агрегат является границей инвариантов и обеспечение транзакционности в рамках одной коллекции документов, кажется достаточным для хранения агрегатов, на практике это не всегда соответствует действительности. Дело в том, что почти в любой предметной области существуют связи между агрегатами (ассоциации). Такие ассоциаций обычно проектируются не с помощью прямых ссылок на другой агрегат, а с помощью объектов-значений, содержащих идентификатор другого агрегата [6]. Обеспечение целостности данных такой модели существенно затрудняется при использовании баз данных, не поддерживающих уровни изоляции (Isolation). Однако NoSQL базы данных отлично подходят для извлечения заранее подготовленных (денормализованных) моделей представления за счет высоких показателей производительности и масштабируемости.

Основываясь на данных утверждениях, можно сделать вывод, что применение схожей с CRUD системами архитектуры является сложным, трудно поддерживаемым и плохо масштабируемым решением.

На сегодняшний день наиболее бескомпромиссным и гибким выглядит подход, построенный на основе принципа CQRS (сокр. от англ. Command-Query Responsibility Segregation – «разделение ответственности между командами и запросами»), позволяющий не только спроектировать выразительную и правильную модель, но и организовать эффективное извлечение и сохранение данных при практически неограниченном уровне вертикальной и горизонтальной масштабируемости системы [3, 8].

Основная идея CQRS – использование разных моделей данных и инфраструктурных механизмов для операций чтения записи данных. В основе данного принципа лежат два понятия: команда (Command), осуществляющая запись дан-

ных, и запрос (Query), возвращающий данные. Фактически, CQRS разделяет операции CRUD на две независимые группы: Read и Create/Update/Delete и представляет их в виде специфических для рассматриваемой предметной области понятий. Схемы такой архитектуры представлена на рисунке 2.

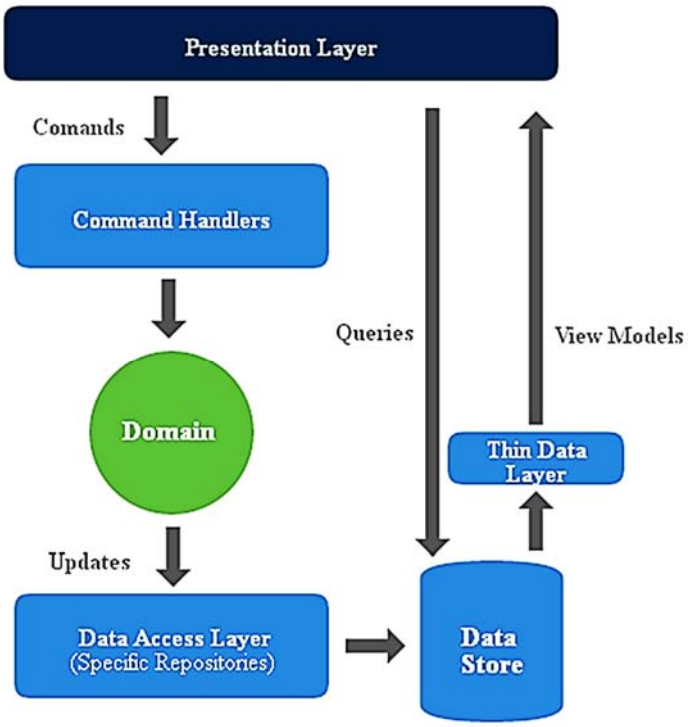

Рис. 2

Принцип CQRS позволяет масштабировать стороны чтения и записи независимо друг другу и проектировать каждую из сторон на основе наиболее подходящих для нее технологий.

Рассматривая CQRS в контексте описанных проблем можно выделить следующее:

1. CQRS значительно облегчает проектирование уровня представления, т.к. позволяет реализовать эффективное и хорошо масштабируемое решение для извлечения сложных структур данных [3, с. 32].

2. CQRS концептуально основывается не на общих операциях чтения/записи, а на командах, специфических для рассматриваемой предметной области и являющихся частью модели, делая ее более простой и выразительной.

3. CQRS предоставляет полную свободу в проектировании инфраструктуры выполнения запросов и команд, делая возможным использование наиболее подходящих технологий для решения каждой из задач. Данный подход делает возможным использование NoSQL базы данных для выполнения запросов к заранее подготовленным, денормализованным данным и реляционную базу данных вместе с технологией ORM для выполнения команд над агрегатами, используя все преимущества ACID. Еще одним вариантом, может быть применение технологии Event Sourcing, предоставляющую еще большую свободу при проектировании системы [3, с. 50].

Принцип CQRS не решает всех проблем проектирования. Он направлен на решение концептуальных проблем и не может обойти все технические ограничения. Однако, разделение ответственности между операциями чтения и записи, позволяет свести эти ограничения к минимуму и использовать «правильные инструменты» для решения проблем.

Следует отметить, что рассмотренные проблемы и пути их решения применимы не только системам, работающим со сложными предметными областями, но и к системам, базирующихся на операциях CRUD. DDD позволяет строить легко поддерживаемые и изящные архитектуры, независимо от сложности рассматриваемой предметной области.

Список литературы

1. Эванс, Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Э. Эванс – Вильямс, 2011. – 448 с.

2. Фаулер, М. Шаблоны корпоративных приложений / М. Фаулер, J.P. – Вильямс, 2011. – Vol. 544.

3. Command and Query Responsibility Segregation (CQRS) Pattern // [Electronic resource] – Mode of access: http://msdn.microsoft.com/en-us/library/dn568103.aspx – Date of access: 15.11.2014.

4. DDD Aggregate Component pattern in action // [Electronic resource] / J. Bogard – Mode of access: http://lostechies.com/jimmybogard/2009/02/05/dddaggregate-component-pattern-in-action – Date of access: 06.08.2014.

5. DDD: Persisting Aggregate Roots In A Unit Of Work// [Electronic resource] / M. Mogosanu – Mode of access: http://www.sapiensworks.com/blog/post/2013/05/01/DDD-Persisting-Aggregate-Roots-In-A-Unit-Of-Work.aspx – Date of access: 11.11.2014.

6. Don't Use ORM Entities To Model The Domain // [Electronic resource] / M. Mogosanu – Mode of access: http://www.sapiensworks.com/blog/post/2012/04/20/Dont-Use-ORM-Entities-To-Model-The-Domain.aspx – Date of access: 09.08.2014.

7. Vernon, V. Implementing Domain-Driven Design / V. Vernon – СПб.: Инкарт, 2013. – Vol. 656.

8. Young G. CQRS Documents by Greg Young / G. Young – Vol 56.