
Оглавление

Предисловие	13
Введение	15
Для кого предназначена эта книга	17
Что я узнаю?	18
Структура книги	18
Условные обозначения	19
Благодарности	20
От издательства	21
Глава 1. Проблемы при управлении данными.....	22
Управление данными	23
Аналитика фрагментирует ландшафт данных.....	26
Скорость доставки программного обеспечения меняется	28
Сети становятся быстрее	29
Приоритет — вопросы конфиденциальности и безопасности	29
Необходимо интегрировать операционные системы и системы обработки транзакций	30
Для монетизации данных требуется экосистемная архитектура	31
Предприятия обременены устаревшими архитектурами данных	32
Корпоративное хранилище данных и бизнес-аналитика	32
Озеро данных	35
Централизованное представление	38
Итоги главы	38
Масштабируемая архитектура	39
Глава 2. Введение в масштабируемую архитектуру:	
организация данных в масштабе	40
Общепризнанные отправные точки	41
У каждого приложения есть база данных	41
Приложения специфичны и обладают уникальным контекстом	41

Золотой источник	42
Дileммы интеграции данных не избежать	42
Приложения играют роли поставщиков и потребителей данных	43
Основные теоретические соображения	44
Принципы объектно-ориентированного программирования	44
Предметно-ориентированное проектирование	46
Бизнес-архитектура	49
Шаблоны связи и интеграции	57
Двухточечная связь	57
Разрозненные хранилища	58
Звездообразная модель	59
Масштабируемая архитектура	60
Золотые источники и хранилища данных предметной области	60
Контракты на поставку данных и соглашения о совместном их использовании	63
Устранение разрозненного подхода	64
Предметно-ориентированное проектирование в масштабе предприятия	65
Оптимизация данных для чтения	68
Уровень данных как целостная картина	69
Метаданные и целевая операционная модель	73
Итоги главы	74
 Глава 3. Управление большими объемами данных:	
архитектура хранилищ данных только для чтения	76
Знакомство с архитектурой RDS	76
Разделение ответственности команд и запросов	77
Что такое CQRS	77
CQRS в масштабе	80
Службы и компоненты хранилища данных только для чтения	85
Метаданные	85
Качество данных	89
Уровни RDS	90
Получение данных	92
Интеграция готовых коммерческих решений	95
Извлечение данных из внешних API и SaaS	96
Служба исторических данных	97
Измерения только для добавления	100
Варианты проектирования	101

Репликация данных	103
Уровень доступа	104
Служба обработки файлов	106
Служба уведомлений о доставке	106
Служба удаления персональной информации	107
Распределенная оркестрация	107
Интеллектуальные службы потребления	108
Заполнение RDS по запросу	111
Рекомендации по использованию RDS	112
Итоги главы	113
 Глава 4. Управление сервисами и API: архитектура API	115
Знакомство с архитектурой API	115
Что такое сервис-ориентированная архитектура	116
Интеграция корпоративных приложений	120
Оркестрация сервисов	122
Хореография сервисов	125
Публичные и частные сервисы	126
Модели сервисов и канонические модели данных	127
Сходства между SOA и архитектурой корпоративного хранилища данных	128
Современный взгляд на SOA	130
API-шлюз	130
Модель ответственности	132
Новая роль ESB	134
Контракты на обслуживание	135
Обнаружение сервисов	135
Микросервисы	136
Роль API-шлюза в микросервисах	138
Функции	139
Сервисная сетка	140
Границы микросервисов	142
Микросервисы в эталонной архитектуре API	142
Коммуникация между экосистемами	143
Каналы связи на основе API	145
GraphQL	147
Метаданные	148

Использование RDS для чтения в реальном времени и активного чтения	149
Итоги главы	150
Глава 5. Управление событиями и ответами: потоковая архитектура.....	152
Знакомство с потоковой архитектурой	152
Асинхронная модель событий имеет значение.....	153
Как выглядят событийно-ориентированные архитектуры.....	154
Топология посредника.....	155
Топология брокера	156
Стили обработки событий	157
Введение в Apache Kafka.....	158
Распределенные события.....	161
Возможности Apache Kafka	162
Потоковая архитектура	163
Производители событий	163
Потребители событий	166
Платформа событий	168
Источники событий и команд	169
Модель управления	172
Бизнес-потоки	173
Шаблоны потребления потоковой передачи.....	176
Передача состояния с помощью событий	178
Роли RDS	179
Использование потоковой передачи для заполнения RDS.....	180
Элементы управления и политики для управления доменами	180
Потоковая передача как операционный конвейер	181
Гарантии и согласованность.....	182
Уровень согласованности	182
Семантики обработки «хотя бы один раз», «точно один раз» и «не больше одного раза».....	183
Порядок сообщений	183
Очередь недоставленных сообщений	183
Потоковое взаимодействие.....	184
Метаданные для моделей управления и самообслуживания	185
Итоги главы	186

Глава 6. Соединение точек.....	188
Кратко об архитектурах.....	188
Архитектура RDS	189
Архитектура API	190
Архитектура потоковой передачи	190
Усиливающие шаблоны	191
Стандарты корпоративной совместимости	193
Конечные точки устойчивых данных	193
Контракты на доставку данных	196
Доступные и адресуемые данные.....	198
Принципы пересечения сетей.....	198
Стандарты корпоративных данных.....	204
Принципы оптимизации потребления	205
Возможность обнаружения метаданных.....	208
Семантическая согласованность	212
Предоставление соответствующих метаданных.....	216
Происхождение и перемещение данных.....	217
Эталонная архитектура	219
Итоги главы	221
 Глава 7. Управление данными и их безопасность.....	223
Управление данными	223
Организация: роли в управлении данными.....	225
Процессы: деятельность по управлению данными.....	228
Люди: доверительные и этические, социальные и экономические соображения	229
Технологии: золотой источник, владение приложениями и их администрирование	230
Данные: золотые источники, золотые наборы данных и классификации	232
Безопасность данных	239
Текущий разрозненный подход.....	240
Единая защита данных для архитектур	241
Поставщики удостоверений.....	243
Эталонная архитектура безопасности и подход к контексту данных	244
Процесс безопасности	246
Практическое руководство	250
Архитектура RDS	250

Архитектура API	252
Потоковая архитектура	256
Интеллектуальный механизм обучения	258
Итоги главы	259
Глава 8. Превращение данных в ценность	260
Модели потребления.....	261
Прямое использование хранилищ данных только для чтения.....	261
Хранилища данных предметной области	262
Целевая операционная модель.....	264
Специалисты по данным как целевая группа пользователей.....	265
Бизнес-требования.....	267
Нефункциональные требования.....	267
Построение конвейера данных и модели данных.....	269
Распространение интегрированных данных	277
Возможности бизнес-аналитики	278
Возможности самообслуживания	280
Возможности аналитики	283
Стандартная инфраструктура для автоматизированного развертывания.....	284
Модели без сохранения состояния.....	285
Предварительно настроенные рабочие места.....	285
Стандартизация шаблонов интеграции моделей.....	286
Автоматизация.....	286
Метаданные модели.....	287
Эталонная архитектура продвинутой аналитики	288
Итоги главы	292
Глава 9. Управление основными корпоративными данными	293
Демистификация управления мастер-данными	294
Стили управления основными данными	294
Эталонная архитектура MDM	297
Разработка решения для управления основными данными	299
Распространение MDM.....	300
Основные идентифицирующие номера	300
Справочные и основные данные	302
Определение области видимости корпоративных данных	302
MDM и качество данных как услуга	305

Курируемые данные.....	305
Обмен метаданными.....	306
Интегрированные представления.....	307
Повторно используемые компоненты и логика интеграции	307
Повторная публикация данных	308
Связь с управлением данными	309
Итоги главы	309
 Глава 10. Демократизация данных с помощью метаданных.....	 311
Управление метаданными.....	312
Модель метаданных предприятия	313
Граф корпоративных знаний.....	321
Архитектурные подходы к управлению метаданными	325
Совместимость метаданных	326
Хранилища метаданных	328
Площадка для быстрого доступа к авторизованным данным.....	331
Итоги главы	334
 Глава 11. Заключение	 336
Модель доставки	337
Полностью децентрализованный подход.....	338
Частично децентрализованный подход	339
Структурирование команд.....	339
Стратегия InnerSource.....	340
Культура	341
Выбор технологий.....	342
Упадок традиционной архитектуры предприятия	343
Чертежи и схемы	344
Современные навыки.....	344
Контроль и управление	344
Послесловие	345
 Глоссарий	 346
Об авторе	366
Об обложке	367

ГЛАВА 2

Введение в масштабируемую архитектуру: организация данных в масштабе

Какая архитектура нужна предприятию, чтобы управление им основывалось на данных? Как эффективно распределять данные, сохраняя гибкость, безопасность и контроль? В этой главе мы рассмотрим эти вопросы и заложим основу для управления данными.

Современные тренды толкают нас на переосмысление способов управления данными и их интеграции. Ранее мы разобрали тесную связь, возникающую при создании точных копий данных, и трудности практического анализа необработанных данных. Мы также обсудили проблемы унификации и огромные усилия, которые прилагаются к созданию интегрированного хранилища данных, и их влияние на гибкость. Нам необходимо перейти от объединения всех данных в одном хранилище к подходу, который позволяет предприятиям, командам и пользователям легко и безопасно распределять, собирать и использовать данные. Платформы, процессы и шаблоны должны упрощать работу другим. Нам нужны простые, хорошо документированные, быстрые и легкие в использовании интерфейсы. Нам нужна масштабируемая архитектура управления данными. Именно эти вопросы мы и затронем в главе. А начнем мы с того, как организовать ландшафт и интегрировать данные.

Крупномасштабная архитектура в моем представлении ориентирована на управление данными и их интеграцию. Это архитектура для предприятий, которая позволяет командам безопасно и легко предоставлять данные, сохраняя гибкость и контроль. Как и во многих других архитектурах, в ней используются *архитектурные строительные блоки*, которые относятся к «пакету функций, определенных для удовлетворения потребностей бизнеса»¹. Эти блоки будут

¹ Согласно OpenGroup (<https://oreil.ly/YXEbm>) способ объединения функциональности, продуктов и нестандартных разработок в архитектурные блоки зависит от архитектуры.

использоваться снова и снова, чтобы помочь вам понять, какую именно часть архитектуры мы обсуждаем.

Начнем с традиционных принципов определения наиболее важных архитектурных блоков. Далее мы обсудим дополнительную литературу и сделаем выводы. Затем, наконец, рассмотрим новую архитектуру и ее обоснование, а также раскроем, что находится внутри нее. К концу этой главы вы поймете, как различные архитектурные блоки работают и связаны между собой. Эта базовая теория необходима для понимания основных движущих сил новой архитектуры. В следующих главах мы подробнее рассмотрим шаблоны, проекты, диаграммы и рабочие процессы, и вы начнете понимать, как эта архитектура связывает воедино все области управления данными.

Общепризнанные отправные точки

Прежде чем погрузиться в обсуждение, я хочу выделить главные отправные точки. Они формируют архитектуру и содержат элементы, на которые я часто ссылаюсь.

У каждого приложения есть база данных

Приложения тесно связаны с базами данных. В контексте управления данными и их перемещения между приложениями мы можем допустить, что у каждого приложения всегда есть база данных. То есть всякий раз, сталкиваясь с необходимостью потребления данных, вы должны хранить их в БД приложения. Да, бывают приложения, хранящие свои данные в другом месте, а еще приложения, использующие одну и ту же базу данных, или приложения, выполняющие обработку в памяти. Но всем этим приложениям все равно нужно где-то хранить свои данные. Так что у них всегда есть хранилище в той или иной форме, а значит, и хранилище данных приложения.

Приложения специфичны и обладают уникальным контекстом

В главе 1 я отметил, что приложения используются для решения *конкретных* задач. Данные каждого приложения уникальны. Существует несколько этапов проектирования и разработки приложений. Все начинается с определения концепции и разработки проекта; затем мы трансформируем наши знания в логическую модель данных, определяющую абстрактную структуру концептуальной информации и требований. Наконец, мы создаем физическую модель данных приложения: истинный проект приложения и базы данных. Физическая модель данных уникальна и принимает как контекстные, так и нефункциональные требования к тому, как приложение и БД будут спроектированы и использованы.

Золотой источник

В фрагментированной и распределенной среде порой сложно определить авторитетные источники исходных и уникальных данных. Поэтому важно знать, откуда поступают данные и где ими управляют. В этой книге я обсуждаю основополагающие концепции золотого источника и золотого набора данных¹.

- *Золотой источник* — это авторитетное *приложение*, в котором все аутентичные данные обрабатываются в определенном контексте. Он состоит из одного или нескольких золотых наборов данных².
- *Золотой набор данных* — это созданные надежные оригинальные *данные*. Он подлинный и уникальный и должен быть точным, полным и известным³. Золотой набор данных состоит из элементов данных (<https://oreil.ly/RGKhT>) — элементарных единиц удобочитаемой информации, имеющих точное значение или точную семантику. У них есть определяемое имя, и они служат связующим звеном для других субъектов управления данными, таких как распоряжение данными.

Благодаря золотому источнику и золотому набору данных вы всегда сможете точно и последовательно отчитаться о своих данных. Это важно для надежного управления данными, о чем мы поговорим позже, в главе 7.

Дileммы интеграции данных не избежать

Для перемещения данных между приложениями их всегда приходится интегрировать. Это происходит из-за уникального контекста, в котором эти данные создаются. Неважно, что вы выполняете — ETL (extract, transform, and load — «извлечение, преобразование и загрузка») или ELT (extract, load, and transform — «извлечение, загрузка и преобразование»), виртуальным или физическим способом, пакетами или в реальном времени: от дилеммы интеграции данных никуда не деться. Преобразование всегда требуется при переносе данных из одного контекста в другой. Ключевое слово *всегда* образует новую архитектуру.

¹ W3C определяет набор данных (<https://oreil.ly/nvHnG>) как «коллекцию данных, которые можно получить или загрузить в одном или нескольких форматах. Неточное определение набора данных было сделано намеренно, чтобы созданный концептуальный набор данных можно было использовать в различных контекстах».

² Поиск надежных приложений порой непрост. Некоторые надежные источники могут быть скрыты за сложными шаблонами, которые необходимо понять, прежде чем найти, что вам нужно. Для получения дополнительной информации о золотых источниках см.: *Graham A. Mastering Your Data* (Koios, 2015).

³ Бывают исключительные ситуации, когда золотые наборы данных не управляются системой, которая фактически создала исходный фрагмент данных.

Приложения играют роли поставщиков и потребителей данных

Приложения выступают в роли либо поставщиков, либо потребителей данных, а иногда, как мы увидим, и в той, и в другой одновременно. Одно приложение использует данные, а другое создает и предоставляет их. В контексте интеграции данных во множество приложений и систем это важно понимать. Роли поставщика и потребителя данных станут строительными блоками формальной архитектуры.

В зависимости от того, действует ли система или приложение в качестве поставщика или потребителя, правила игры меняются: применяются разные архитектурные принципы (рис. 2.1).



Рис. 2.1. Роли приложения как поставщика и потребителя данных будут формировать нашу архитектуру

Роли поставщика и потребителя данных также применимы к *внешним сторонам*. Внешними называют стороны, действующие за пределами логических границ экосистемы предприятия. Обычно они находятся в отдельных неконтролируемых местах в сети. В главе 1 мы говорили, что компании рассматривают общедоступную сеть как кладезь (открытых) данных, которые они могут монетизировать для создания услуг и обмена ими. Этот новый способ сотрудничества изменил границы организаций. Новая архитектура, которую я представляю, требует гибкости, чтобы внешние стороны могли быть как поставщиками, так и потребителями данных. Обычно ради этого приходится применять дополнительные меры безопасности, ведь внешние стороны не всегда надежны или известны. Иногда они напрямую связаны с более широким контекстом.

Уникальный контекст, в котором работают приложения, роли поставщика и потребителя данных, а также преобразование, которое всегда происходит между приложениями, будут определять новую архитектуру. Но что еще нужно учитывать? Что делает общую архитектуру хорошей? В следующих разделах мы исследуем эти вопросы.

Основные теоретические соображения

Интеграция данных между приложениями — это управление сложностью взаимодействия и согласования данных между системами. В архитектуре предприятия мы обычно рассматриваем более широкую картину. Но как компоненты работают вместе для интеграции данных внутри приложения и что мы можем из этого извлечь?¹

Интеграция приложений находится на уровне архитектуры или разработки программного обеспечения. На абстрактном уровне интеграция корпоративных данных и интеграция программного обеспечения близки, а порой и частично совпадают (рис. 2.2).



Рис. 2.2. Дисциплины интеграции данных и интеграции программного обеспечения имеют множество пересечений

Далее мы внимательно рассмотрим несколько шаблонов разработки программного обеспечения, позволяющих разобраться в повторяющихся проблемах. Их понимание помогает командам разработчиков избежать создания зависимостей и сложностей.

Принципы объектно-ориентированного программирования

Упомянутые ниже элементы — это те принципы объектно-ориентированного проектирования, которые часто используются для создания независимых программных компонентов. Они все еще применяются в современных популярных методах.

¹ Архитектура предприятия (enterprise architecture, EA) — это план воплощения стратегии предприятия (бизнес-целей и задач) в успешные изменения. Она логически фокусируется на целом предприятии, включая бизнес, информацию (данные), приложения, безопасность и инфраструктуру, тогда как архитектура данных в первую очередь ориентирована на данные.

Первый принцип, который мы рассмотрим, — это *объектно-ориентированное программирование* (ООП). Управление сложностью приложения и кода в целом осуществляется путем абстрагирования сложной логики, реализации общих функций для выполнения повторяющихся задач и создания общих интерфейсов для других компонентов.

В брошюре Роберта К. Мартина (Robert C. Martin) *Design Principles and Design Patterns* (object-mentor.com, 2000) изложены идеи, идеально подходящие для интеграции данных. Многие из его принципов проектирования до сих пор применяются в стандартной практике.

- *Принцип единственной ответственности.* Этот принцип заключается в задании четких обязанностей и границ. Как пишет Мартин, «этот принцип — для людей. Нужно изолировать модули от сложностей организации в целом и спроектировать системы так, чтобы каждый модуль отвечал за потребности только одной бизнес-функции»¹.
- *Принцип инверсии (внедрения) зависимостей.* Он гласит: «Модули высокого уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракций. Абстракции не должны зависеть от деталей — детали должны зависеть от абстракций»². Речь идет о сокрытии внутренней сложности и деталей. Абстракция более стабильна, чем лежащая в основе логика.
- *Принцип открытости/закрытости.* Этот принцип гласит, что «модуль должен быть открыт для расширения, но закрыт для модификации»³. Когда мы изменяем функцию, все, что зависит от нее, также должно быть изменено. Нельзя, чтобы изменения в обмене данными вызывали каскад последующих изменений в зависимых системах или приложениях.
- *Принцип устойчивых зависимостей.* Принцип гласит, что «программные модули зависят от направления устойчивости. Устойчивость связана с объемом работы, необходимой для внесения изменений»⁴. Функциональность приложения должна основываться только на функциях, которые по крайней мере столь же устойчивы, как и модуль.
- *Принцип устойчивой абстракции.* Он гласит, что «компонент должен быть настолько абстрактным, насколько и стабильным»⁵. Преимущество абстрактных стабильных компонентов в том, что вы их легко расширять, не нарушая проект.

¹ Мартин Р. К. и др. Быстрая разработка программ. Принципы, примеры, практика.

² Там же.

³ Martin R. C. The Open-Closed Principle, C++ Report. 1996. <https://oreil.ly/dIJgo>.

⁴ Мартин Р. К. и др. Быстрая разработка программ. Принципы, примеры, практика.

⁵ Martin R. C. OO Design Quality Metrics. 1994. <https://oreil.ly/jl8Cq>.

Новая архитектура была вдохновлена именно этими принципами Мартина, потому что способ взаимодействия приложений через интерфейсы напоминает способ взаимодействия компонентов внутри приложения. Чтобы избежать нарушений в работе приложений и систем при каждом изменении интерфейса, мы должны действовать в соответствии с принципами Мартина. Они пользуются большим уважением и повлияли на многие структуры и методологии разработки. Одна из них называется *предметно-ориентированным проектированием* (domain-driven design, DDD).

Предметно-ориентированное проектирование

Предметно-ориентированное проектирование — это подход к разработке программного обеспечения, который включает сложные системы для крупных организаций, первоначально описанный Эриком Эвансом (Eric Evans)¹. Принцип DDD популярен, потому что многие из его высокоуровневых практик оказали влияние на современные подходы к разработке ПО и приложений, например микросервисы.

Ограниченный контекст

Один из шаблонов предметно-ориентированного проектирования называется *ограниченным контекстом*. Ограничные контексты используются для установки логических границ пространства решений предметной области, чтобы лучше управлять сложностью. Важно, чтобы команды понимали, какие аспекты, включая данные, они могут изменять самостоятельно, а какие являются общими зависимостями, изменения в которых необходимо согласовывать с другими командами, чтобы ничего не сломать. Границы помогают командам и разработчикам более эффективно управлять зависимостями.

Логические границы, как правило, явные и применяются в областях с четкой и более выраженной связностью. Эти зависимости предметной области могут располагаться на разных уровнях, таких как определенные части приложения, процессы, связанные структуры баз данных и т. д. Ограничный контекст является полиморфным и может применяться ко многим различным точкам зрения. *Полиморфизм* означает возможность изменения размера и формы ограниченного контекста в зависимости от точки зрения и окружения. Это также означает, что ограниченный контекст нужно использовать явно; иначе цель его применения останется довольно неясной.

¹ Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.

ПРЕДМЕТНЫЕ ОБЛАСТИ И ОГРАНИЧЕННЫЕ КОНТЕКСТЫ

DDD различает ограниченные контексты, предметные области и подобласти. *Предметные области* — это зоны проблем, которые мы пытаемся решить; те области, в которых сочетаются знания, поведение, законы и действия; те области, в которых мы видим семантическую связь: поведенческие зависимости между компонентами или службами. Предметные области обычно разбиваются на подобласти ради более эффективного управления сложностью. Типичный пример — разбиение предметных областей так, чтобы каждая подобласть соответствовала отдельному подразделению организации.

Не все подобласти одинаковы. Их можно разделить на основные, общие или вспомогательные. *Основные подобласти* являются наиболее важными. Это секретный ингредиент, который делает бизнес уникальным. *Общие подобласти* неспецифичны и обычно легко управляются с помощью готовых продуктов. *Вспомогательные подобласти* не дают конкурентных преимуществ, но необходимы для работы организации. Обычно они довольно простые.

Ограниченные контексты — это логические (контекстные) границы. Они сосредоточены на пространстве решений: проектировании систем и приложений. Именно в этом месте разумнее всего сфокусироваться на пространстве решений. Это может быть код, дизайн базы данных и т. д. Области и ограниченные контексты могут совпадать, но их не обязательно связывать. Ограничные контексты носят технический характер и поэтому могут охватывать несколько предметных областей и подобластей.

Идея заключается в том, что после задания логических границ зоны ответственности становятся более явными и управляемыми. Общение между членами команды становится эффективнее, ведь все они работают над схожими задачами. Установление логических границ ПО похоже на принцип *единственной ответственности*, который гласит: «Что принадлежит одной области, должно оставаться (и эффективно управляться) в этой же области».

На уровне предприятия мы логически группируем приложения с высокой степенью связности или общими интересами. Общие интересы обычно находятся на уровне связности задач и ответственности бизнеса. Некоторые называют их функциональными областями, логическими группами, кластерами или организационными возможностями. Идея группировки сущностей, процессов или приложений не нова.

Модель предметно-ориентированного проектирования отличается от «традиционной» группировки *строгими границами* DDD. Эванс утверждает, что ограниченные контексты могут развиваться независимо, но должны быть разделены. Разделение обычно осуществляется через сокрытие сложных внутренних функций приложения и обеспечение определенного уровня устойчивости интерфейсов и уровней. Эти принципы аналогичны принципам *инверсии зависимостей* и *устойчивых зависимостей*.

Единый язык

«Единый язык — термин, который Эрик Эванс использует в предметно-ориентированном проектировании для описания практики построения общего строгого языка для общения разработчиков и пользователей», — сказал Мартин Фаулер (Martin Fowler) (<https://oreil.ly/enG7A>). Этот язык похож на определения, лексику или специализированную терминологию специалистов конкретной индустрии. Единый язык помогает сплотить людей в более крупных командах, потому что в большой команде или на предприятии часто бывает непросто прийти к взаимопониманию по вопросу языка. Чтобы избежать чрезмерного перекрестного общения и несостыковок в терминологии, для поддержки разработки приложений или программного обеспечения в предметной области используются единые языки. Унифицированного языка не существует, хотя между ними могут быть совпадения.



Я настоятельно рекомендую прочесть книгу *Semantic Software Design* Эбена Хьюитта (Eben Hewitt) (O'Reilly, 2019). В ней используется методология конструктивного мышления и проводится параллель между семантикой и проектированием архитектуры.

Ограниченный контекст и единый язык сильно взаимосвязаны, поскольку в DDD ожидается, что каждый ограниченный контекст будет иметь свой единый язык. Приложения или его компоненты, принадлежащие друг другу и управляемые в ограниченном контексте, должны использовать одинаковый язык. Если ограниченный контекст растет и нет взаимопонимания между членами команды, контекст может быть разбит на более мелкие части. Если ограниченный контекст изменится, ожидается, что и единый язык будет другим. Эмпирическое правило заключается в том, что один ограниченный контекст управляет одной командой (гибкой разработки или DevOps), потому что членам одной команды легче понимать текущую ситуацию и все ее зависимости.

Говоря об ограниченном контексте, я часто сравниваю его с культурой, чтобы показать, что в разных командах используются разные определения и терминология. Контекст специфичен и обычно основан на наших знаниях. У каждого ограниченного контекста своя цель, свой фон и, что наиболее важно, своя культура. Культура во многом определяет, как мы думаем, проектируем и моделируем. Внутри культуры есть субкультуры: группы людей в рамках более широкой культуры, которых объединяет общий набор целей и практик. То же можно сказать о предметно-ориентированном проектировании. Ограниченный контекст может состоять из нескольких подобластей или нескольких ограниченных контекстов. Точно так же в главе 1 мы говорили, что проект и модель данных приложения получают контекст из области, в которой они используются. Все эти подходы к проектированию тесно связаны.

Подход к предметно-ориентированному проектированию использует ограниченные контексты для установления границ независимых областей с более высоким уровнем связности. Если мы спроектируем DDD на нашу среду приложений на уровне предприятия, то границы области будут удерживать вместе не только приложение, но и язык, знания, ресурсы команды и технологии. Как показано на рис. 2.3, мы ожидаем, что каждый ограниченный контекст будет иметь свои приложение, данные, процессы и определения контекста (единий язык).



Рис. 2.3. DDD делит сложный ландшафт на ограниченные контексты и защищает границы между ними. В этом примере приложения А и Б имеют собственный ограниченный контекст

Недостатком использования приложений в качестве границ является то, что это все еще звучит очень размыто, если мы говорим о предприятии. Что объединяет элементы и компоненты в большую архитектуру? Обычно проблема заключается в том, как проектируются и разрабатываются приложения для бизнеса. Это подводит нас к следующей методологии: бизнес-архитектуре.

Бизнес-архитектура

Хорошо спроектированная *бизнес-архитектура* имеет решающее значение для успешного построения архитектуры предприятия. Гильдия бизнес-архитектуры описывает ее как «целостное, многомерное бизнес-представление о: возможностях, комплексной доставке ценностей, информации и организационной структуре, а также отношения между этими бизнес-взглядами и стратегиями, продуктами, политиками, инициативами и заинтересованными сторонами»¹. Архитекторы используют бизнес-архитектуру как основу для определения принципов, рекомендаций, желаемых результатов и границ предприятия и его бизнес-экосистемы. Бизнес-архитектура поддерживается путем построения целостных и многомерных бизнес-представлений с бизнес-возможностями². Мы можем

¹ Гильдия бизнес-архитектуры, статья: A Guide to the Business Architecture Body of Knowledge 7.5. – 2019. – С. 2. <https://oreil.ly/b3db5>.

² Определение возможности было первоначально придумано Ульрихом Хоманном (Ulrich Homann) в статье: A Business-Oriented Foundation for Service Orientation. 2006. Гильдия бизнес-архитектуры заимствовала его в проекте Business Architecture Body of Knowledge.

проводить параллель между ограниченным контекстом и бизнес-архитектурой или бизнес-возможностями. Еще один способ использования ограниченного контекста — посмотреть на него через призму бизнес-архитектуры.

Бизнес-возможности

Для решения бизнес-задач и удовлетворения потребностей часто используются бизнес-возможности, включающие создание объектов для каждой бизнес-цели. Бизнес-возможности — это строительный блок, используемый в бизнес-архитектуре. По словам Ульриха Хоманна (Ulrich Homann), бизнес-возможности — это «особые способности или возможности, которыми бизнес может обладать или обмениваться для достижения определенных результатов»¹. Бизнес-возможности — абстракция бизнес-реальности для помощи компаниям в достижении своих стратегических бизнес-целей и стремлений. Они фиксируют и описывают взаимосвязь между данными, процессами, организацией и технологиями в определенном контексте.



Модель бизнес-возможностей представляет общие стратегические бизнес-цели и действия организации в структурированном виде. Каждая бизнес-возможность реализуется как минимум единожды. Эту модель также можно использовать для сопоставления и построения графиков зависимостей. Например, сопоставление основных показателей эффективности управления данными с реализованными бизнес-возможностями показывает эффективность управления данными и его влияние на организацию.

Ограничные контексты, которые используются для задания логических границ, могут быть согласованы с бизнес-архитектурой. На самом высоком концептуальном уровне мы сопоставляем все стратегические бизнес-цели с бизнес-возможностями и группируем их вместе в бизнес-возможности и *потоки создания ценности*². Макет, более конкретная архитектура, создается уровнем ниже в архитектуре приложения. В этом проекте можно провести логические границы (решения), рассматриваемые как ограниченный контекст, представляющий функциональную и прикладную часть бизнеса и реализацию бизнес-архитектуры. Приложения и их компоненты используются для реализации конкретного *технологического* аспекта бизнес-возможностей.

¹ Крис Ричардсон (Chris Richardson), один из авторов Microservices.io, также признал точку зрения бизнес-возможностей (<https://oreil.ly/wBZj2>).

² Потоки создания ценности (<https://oreil.ly/ZgjuE>) — артефакты в рамках бизнес-архитектуры, которые позволяют бизнесу определять ценностное предложение, полученное от внешней (например, покупателя) или внутренней заинтересованной стороны организации.

ГРАНИЦЫ ПРЕДМЕТНОЙ ОБЛАСТИ И ДЕТАЛИЗАЦИЯ

Установка точных границ и детализации — неточная наука. Это мастерство, которое приходит с практикой. Для целей управления данными и понимания масштаба и сложности предприятия я предпочитаю проводить границы предметной области по логическим границам бизнес-архитектуры. Другие больше обращают внимание на организационные границы, бизнес-процессы или знания в предметной области. Еще один способ разграничения областей — разработать подробный план архитектуры программного обеспечения и определить, какие из компонентов приложения должны быть связаны сильнее или слабее. Этот метод особенно популярен у разработчиков микросервисов. Все эти подходы допустимы, и конкретный выбор будет зависеть только от контекста.

Чтобы лучше понять, что такое границы предметной области и как установить ограниченные контексты, рекомендую вам ознакомиться с историями о предметных областях (<https://oreil.ly/Imivx>), со штурмом событий (<https://oreil.ly/DH4OJ>) и с холстом ограниченного контекста (<https://oreil.ly/cWXvT>). Все эти методы предназначены для понимания сложности предметной области, изучения происходящего внутри нее и изучения возможностей ее структурирования или декомпозиции.

Бизнес-возможности в бизнес-архитектуре остаются абстрактными и могут быть реализованы несколько раз. При создании они называются *экземплярами возможности*. Чтобы помочь вам лучше понять эту концепцию, я сделал организационный пример (рис. 2.4) бизнес-возможностей и соответствующих экземпляров возможностей.

Управление взаимоотношениями с клиентами (customer relationship management, CRM) может быть реализовано в качестве бизнес-возможности как для розничного, так и для корпоративного бизнес-отдела компании. Эта же бизнес-возможность может быть осуществлена централизованно в виде модели обслуживания и предоставлена нескольким отделам. Интегрировать или централизовать — вопрос выбора. Централизованное внедрение CRM предпочтительнее, когда необходим контроль, например, с точки зрения безопасности или соответствия. Внедрение CRM в каждом бизнес-подразделении предпочтительнее при разной динамике или конфликте интересов команд.



В книге *Enterprise Architecture As Strategy* Джини В. Росс (Jeanne W. Ross), Питера Вейля (Peter Weill) и Дэвида С. Робертсона (David C. Robertson) (Harvard Business Press, 2006) упоминаются четыре различные классификации операционных моделей, которые также могут использоваться для бизнес-возможностей.

- *Унификация* — для высокой стандартизации и высокой интеграции.
- *Копирование* — для высокой стандартизации и низкой интеграции.
- *Координация* — для низкой стандартизации и высокой интеграции.
- *Диверсификация* — для низкой стандартизации и низкой интеграции.