Содержание

Глава 3

Карты контекстов

Какой бы курс вы ни избрали, всегда найдется тот, кто скажет вам, что вы не правы. Всегда возникнут сложности, подталкивающие вас к мысли о том, что правы ваши критики. Для того чтобы разработать план действий и следовать ему до конца, нужна храбрость.

Ральф Уолдо Эмерсон

КАРТУ КОНТЕКСТОВ проекта можно представить двумя способами. Проще всего нарисовать простую диаграмму, содержащую отображения между двумя или несколькими существующими **ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ (2)**. Следует, однако, понимать, что в этом случае вы рисуете простую диаграмму того, что уже существует. Рисунок иллюстрирует, как ОГРАНИЧЕННЫЕ КОНТЕКСТЫ реального программного обеспечения в пространстве решения связаны друг с другом посредством интеграции. Это значит, что более подробный способ описания КАРТЫ КОНТЕКСТОВ представляет собой реализацию исходного кода интеграции. В этой главе мы рассмотрим оба способа, но большинство деталей реализации мы все же отложим до рассмотрения **ИНТЕГРАЦИИ ОГРАНИЧЕННЫХ КОНТЕКСТОВ (INTEGRATING BOUNDED CONTEXTS) (13)**.

Имейте в виду, что эта глава посвящена *анализу пространства решений,* в то время как в предыдущей главе мы исследовали *оценку пространства задач.*

Назначение главы

- Объяснить важность создания КАРТЫ КОНТЕКСТОВ для успеха проекта.
- Рассмотреть основные организационные и системные отношения, а также их влияние на проект.
- Научиться на примере команды SaaSOvation разрабатывать КАРТЫ для контроля проекта.

Важность КАРТ КОНТЕКСТОВ

Приступая к проектированию в рамках подхода DDD, сначала нарисуйте визуальную КАРТУ КОНТЕКСТОВ *текущей ситуации вашего проекта*. Нарисуйте КАРТУ КОНТЕКСТОВ, состоящую из текущих ограниченных контекстов вашего проекта, а также интеграционные отношения между ними. На рис. 3.1 показана абстрактная карта контекстов. По мере продвижения вперед мы детализируем ее.

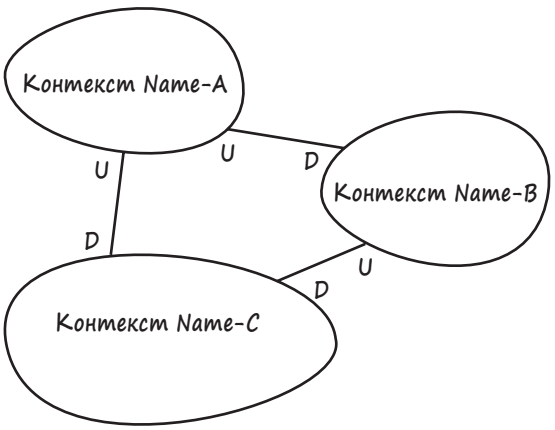


Рис. 3.1. КАРТА КОНТЕКСТОВ абстрактной ПРЕДМЕТНОЙ ОБЛАСТИ. Показаны три ОГРАНИЧЕННЫХ КОНТЕКСТА и отношения между ними. Буква "U" означает отношение "вышестоящий", а "D" — отношение "нижестоящий"

Этот простой рисунок — КАРТА вашей команды. Другие команды проекта могут ссылаться на нее, но при этом, следуя принципам DDD, они обязаны создать собственные КАРТЫ. Основное предназначение вашей карты — дать вашей команде представление о пространстве решений, в котором они пребывают. Другие команды могут не использовать подход DDD и/или не интересоваться вашим мнением.

> **О, нет! Новая терминология!**
>
> Сейчас мы введем термины БОЛЬШОЙ КОМОК ГРЯЗИ (BIG BALL OF MUD), ЗАКАЗЧИК–ПОСТАВЩИК (CUSTOMER-SUPPLIER) и КОНФОРМИСТ (CONFORMIST). Спокойствие! Эти понятия и отношения, связанные с интеграцией, будут рассмотрены в этой главе позднее.

Например, когда вы интегрируете ОГРАНИЧЕННЫЕ КОНТЕКСТЫ в рамках крупного предприятия, вам понадобится интерфейс с **БОЛЬШИМ КОМКОМ ГРЯЗИ**. Команда, поддерживающая целостность "комка грязи", может не интересоваться

направлением вашего проекта, пока вы не присоединитесь к ее интерфейсу прикладного программирования. Ее не интересует ни ваша КАРТА, ни то, что вы будете делать с ее интерфейсом прикладного программирования. И все же ваша КАРТА нужна для отображения отношений с этой командой, потому что *она помогает вашей команде понять ситуацию и обозначает области, в которых необходимо обеспечить взаимодействие между командами*. Это понимание обеспечит успех вашей команды.

Средства коммуникации

Помимо перечня систем, с которыми вы обязаны контактировать, КАРТА КОНТЕКСТОВ служит стимулятором коммуникации между командами.

Представьте себе, что произойдет, если ваша команда будет считать, что команда, поддерживающая целостность "комка грязи", создаст новые интерфейсы прикладного программирования, в то время как она не собирается этого делать или не знает о том, о чем вы думаете. Ваша команда рассчитывает на установление отношения **ЗАКАЗЧИК–ПОСТАВЩИК** с "комком грязи". Однако команда, поддерживающая работу прежней системы, предоставляя только те средства, которые у нее имеются, вынуждает вашу команду установить с ней неожиданное отношение **КОНФОРМИСТ**. В зависимости от того, насколько поздно вы узнаете плохие новости, это невидимое, но реальное отношение может задержать вашу поставку и даже привести к краху проекта. Нарисовав КАРТУ КОНТЕКСТОВ на ранней стадии проекта, вы будете вынуждены осторожно размышлять о ваших отношениях со всеми другими проектами, от которых вы зависите.

> Определите все модели, используемые в проекте, и задайте для каждой свой ОГРАНИЧЕННЫЙ КОНТЕКСТ... Дайте каждому ОГРАНИЧЕННОМУ КОНТЕКСТУ имя и включите эти имена в ЕДИНЫЙ ЯЗЫК проекта. Опишите точки соприкосновения между моделями, явно задавая трансляцию для любого способа коммуникации и выделяя любые совместно используемые ресурсы [Эванс, с. 305].

Приступая к разработке совершенно новой модели, команда CollabOvation должна была использовать КАРТУ КОНТЕКСТОВ. Несмотря на то что модель разрабатывалась практически "с нуля", представление своих предположений о проекте в виде КАРТЫ побудило бы команду подумать об отдельных ОГРАНИЧЕННЫХ КОНТЕКСТАХ. Основные элементы модели следовало бы перечислить на доске, а затем сгруппировать связанные друг с другом лингвистические термины. Это заставило бы команду очертить лингвистические границы и нарисовать простую КАРТУ

КОНТЕКСТОВ. Однако команда не поняла важности стратегического моделирования. Для начала члены команды должны были бы освоить метод стратегического моделирования. Впоследствии они поняли важность этого инструмента, спасающего проект, применив его к общей выгоде. Приступая к разработке СМЫСЛОВОГО ЯДРА проекта, они снова отклонились от курса.

Посмотрим, как можно быстро создать полезную КАРТУ КОНТЕКСТОВ.

Рисование КАРТ КОНТЕКСТОВ

КАРТА КОНТЕКСТОВ описывает *существующую* территорию. Во-первых, мы должны отображать существующее положение дел, а не фантазировать о будущем. Если по мере развития проекта ландшафт изменится, вы сможете обновить КАРТУ. Сначала сосредоточьтесь на текущей ситуации, чтобы понять, где вы находитесь, и определить, куда двигаться дальше.

Создание графических КАРТ КОНТЕКСТОВ не должно быть сложным. Для начала нарисуйте диаграмму маркером на доске. Как указывает Брандолини [Brandolini], стиль "рисуй и стирай" осваивается легко. Если вы решите для рисования диаграмм использовать какой-то инструмент, постарайтесь, чтобы он был неформальным.

Обратите внимание на то, что имена ОГРАНИЧЕННЫХ КОНТЕКСТОВ на рис. 3.1, а также отношения интеграции на диаграмме представляют собой всего лишь пустые поля для заполнения (placeholders). На реальной карте они будут заменены конкретными именами. На диаграмме показаны отношения "вышестоящий" и "нижестоящий", смысл которых будет объяснен в этой главе позднее.

Заметки на доске

Нарисуйте простую диаграмму текущей ситуации, в которой указаны границы и отношения между командами, виды используемой интеграции, а также необходимые трансляции между ними.

> Помните: то, что вы нарисуете, будет реализовано в виде программного обеспечения. Если вам нужна дополнительная информация о том, что вы рисуете, исследуйте систему, с которой интегрируется ваш ОГРАНИЧЕННЫЙ КОНТЕКСТ.

Иногда возникает желание укрупнить диаграмму и добавить детали в конкретную часть КАРТЫ КОНТЕКСТОВ. В этом случае просто возникает другая точка зрения на тот же самый КОНТЕКСТ или те же самые КОНТЕКСТЫ. Помимо

границ, отношений и трансляций, возможно, потребуется включить в диаграмму другие элементы, такие как **МОДУЛИ (9)**, важные **АГРЕГАТЫ (10)**, возможно, места расположения команд, а также другую информацию, относящуюся к данному КОНТЕКСТУ. Этот способ демонстрируется далее в этой главе.

При необходимости все рисунки и любые тексты можно включить в справочник. При этом следует избегать формальностей, чтобы он оставался простым и гибким. Чем более формальными вы будете, тем меньше людей захотят пользоваться вашей КАРТОЙ. Слишком большое количество деталей на диаграммах вряд ли поможет вашей команде. Главное — доверительное общение. Если в ходе разговора вы уточнили стратегическую точку зрения, добавьте его результат в КАРТУ КОНТЕКСТОВ.

Нет, это не АРХИТЕКТУРА ПРЕДПРИЯТИЯ

КАРТА КОНТЕКСТОВ — это *не* АРХИТЕКТУРА ПРЕДПРИЯТИЯ и не диаграмма топологии системы.

Содержащаяся на ней информация относится к взаимодействию моделей и организационных шаблонов DDD. Тем не менее КАРТЫ КОНТЕКСТОВ можно использовать для высокоуровневых архитектурных исследований, если у вас нет других источников информации о структуре предприятия. Кроме того, они помогают выявлять архитектурные недостатки, например узкие места интеграции. Поскольку КАРТЫ КОНТЕКСТОВ отображают организационную динамику, они могут помочь решить даже трудные вопросы управления, блокирующие развитие проекта, а также другие проблемы, возникающие перед командой и руководителями, которые сложно выявить другими методами.

Ковбойская логика

AJ: Жена сказала: “Я была на пастбище с коровами. Ты меня не заметил?” Я ответил: “Не-а”. После этого она неделю со мной не разговаривала.

Диаграммы должны постоянно висеть на стенах в ваших кабинетах. Если члены команды часто посещают корпоративный вики-сайт, то диаграммы следует загрузить и туда. Если корпоративный вики-сайт будет часто игнорироваться, не беда. Как говорится, вики-сайт — это место, куда информацию отправляют умирать. Однако независимо от того, где именно отображаются КАРТЫ КОНТЕКСТОВ, на них не будут обращать внимания, если команда не проводит регулярных и содержательных дискуссий.

Проектные и организационные отношения

Напомним, что компания SaaSOvation планирует разработку и совершенствование трех продуктов.

1. Набор программ для обеспечения корпоративного сотрудничества CollabOvation, позволяющий зарегистрированным пользователям публиковать информацию, имеющую бизнес-ценность, используя популярные веб-инструменты, такие как форумы, общие календари, блоги, базы знаний и т.д. Это основной продукт компании SaaSOvation и ее первое **СМЫСЛОВОЕ ЯДРО (2)** (хотя пока команда не знает терминологии DDD). Это КОНТЕКСТ, из которого в конце концов будет извлечена модель IdOvation (точка 2). В данный момент проект CollabOvation использует модель IdOvation как **НЕСПЕЦИАЛИЗИРОВАННУЮ ПОДОБЛАСТЬ (2)**. Сам проект CollabOvation будет играть роль **СЛУЖЕБНОЙ ПОДОБЛАСТИ (2)** как вспомогательная надстройка программы ProjectOvation (точка 3).
2. Будучи повторно используемой сущностью и моделью управления доступом, модель IdOvation обеспечивает безопасное ролевое управление доступом для зарегистрированных пользователей. Сначала эти свойства сочетались с проектом CollabOvation (точка 1), но эта реализация была ограниченной и не допускала повторного использования. Тогда компания SaaSOvation выполнила рефакторинг программы CollabOvation, создав новый и ясный ОГРАНИЧЕННЫЙ КОНТЕКСТ. Главной функциональной возможностью продукта является поддержка многих арендаторов, что является жизненно важным для приложения SaaS. Модель IdOvation играет роль СЛУЖЕБНОЙ ПОДОБЛАСТИ для моделей, частью которых она является.
3. Продукт для управления гибким проектированием ProjectOvation в данный момент является новым СМЫСЛОВЫМ ЯДРОМ. Пользователи этого продукта компании SaaS могут создавать ресурсы для управления проектом, а также выполнять анализ и проектирование артефактов и отслеживание прогресса с помощью каркаса на основе методологии Scrum. Как и проект CollabOvation, проект ProjectOvation использует модель IdOvation как НЕСПЕЦИАЛИЗИРОВАННУЮ ПОДОБЛАСТЬ. Одна из новаторских функций предусматривает взаимодействие между командами (точка 1) при управлении гибким проектированием, позволяя проводить обсуждение продуктов, выпусков, спринтов и отдельных задач в рамках технологии Scrum.

Итак, определения!

Упомянутые выше проектные и организационные отношения определяются ответами на следующие вопросы.

Какие отношения существуют между ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ и отдельными командами проекта? Существует несколько организационных и интеграционных шаблонов DDD, один из которых почти всегда существует между любыми двумя ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ. Почти все определения, приведенные ниже, взяты из книги [Evans, Ref].

- **ПАРТНЕРСТВО (PARTNERSHIP).** Когда команды в двух КОНТЕКСТАХ достигают успеха или терпят неудачу вместе, должно возникнуть отношение сотрудничества. Эти команды устанавливают процесс координированного планирования разработки и совместное управление интеграцией. Они должны сотрудничать в процессе эволюции своих интерфейсов, чтобы учитывать потребности обеих систем. Взаимозависимые функции должны быть организованы так, чтобы завершение их разработки происходило в рамках одного и того же выпуска.
- **ОБЩЕЕ ЯДРО (SHARED KERNEL).** Общая часть модели и связанного с ней кода образует очень тесную взаимосвязь, которая может как способствовать работе, так и мешать ей. Обозначьте четкую границу определенного подмножества модели предметной области, которую команды согласны считать общей. Ядро должно быть маленьким. Оно имеет особый статус и не должно изменяться без консультаций с другой командой. Установите непрерывный интеграционный процесс, сохраняющий ядро небольшим, и согласуйте **ЕДИНЫЙ ЯЗЫК (1)** команд.
- **РАЗРАБОТКА «ЗАКАЗЧИК–ПОСТАВЩИК» (CUSTOMER–SUPPLIER DEVELOPMENT).** Когда две команды находятся в отношении "нижестоящий–вышестоящий", причем успех вышестоящей команды зависит от нижестоящей команды, потребности нижестоящей команды можно удовлетворить разными способами, имеющими разные последствия. Следует учитывать приоритеты нижестоящих команд при планировании работы вышестоящей команды. Проводите переговоры и согласовывайте сметы, учитывающие потребности нижестоящих команд, чтобы все понимали взятые обязательства и календарный план.
- **КОНФОРМИСТ (CONFORMIST).** Когда две команды находятся в отношении "вышестоящий–нижестоящий", причем вышестоящая команда не имеет причин учитывать потребности нижестоящей команды, то нижестоящая команда беспомощна. Руководствуясь альтруизмом, члены вышестоящей команды могут давать обещания, но они вряд ли будут выполнены. Нижестоящая команда устраняет сложность трансляции между ограниченными контекстами, беспрекословно подчиняясь модели вышестоящей команды.
- **ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ (ANTICORRUPTION LAYER).** При объединении хорошо продуманных ограниченных контекстов с тесно

сотрудничающими командами уровни трансляции могут быть простыми и даже элегантными. Однако если управление или коммуникация не соответствует общему ядру, партнеру или отношению "заказчик–поставщик", то трансляция становится более сложной. Уровень трансляции усиливает защиту. Нижестоящий клиент должен создать изолирующий слой, чтобы обеспечить свою систему функциональной возможностью вышестоящей системы в терминах своей модели предметной области. Этот уровень общается с другой системой с помощью существующего интерфейса, не требуя или почти не требуя модификации другой системы. Внутренне он транслирует информацию в одном или обоих направлениях между моделями.

- **СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ (OPEN HOST SERVICE).** Определите протокол, предоставляющий доступ к вашей системе, как к набору служб. Откройте этот протокол, чтобы его могли использовать все, кто захотят интегрироваться с вами. Уточните и расширьте этот протокол, чтобы учесть новые требования интеграции, за исключением специфических потребностей. Затем используйте одноразовый транслятор для расширения протокола в особых ситуациях, чтобы протокол оставался простым и согласованным.
- **ОБЩЕДОСТУПНЫЙ ЯЗЫК (PUBLISHED LANGUAGE).** Трансляция между моделями двух ОГРАНИЧЕННЫХ КОНТЕКСТОВ требует общего языка. Используйте в качестве основной среды коммуникации хорошо документированный общий язык, который может выразить необходимую информацию о предметной области, выполняя при необходимости перевод информации с другого языка на этот и с этого языка на другой. ОБЩЕДОСТУПНЫЙ ЯЗЫК часто сочетается со СЛУЖБОЙ С ОТКРЫТЫМ ПРОТОКОЛОМ.
- **ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ (SEPARATE WAYS).** Мы должны быть строгими при определении требований. Если между двумя наборами функциональных возможностей нет важного отношения, их можно полностью отсоединить друг от друга. Интеграция всегда дорого стоит, а выгоды иногда бывают незначительными. Объявите ограниченный контекст, не имеющий связей с другими контекстами. Это позволит разработчикам найти простые и специальные решения в этой небольшой области.
- **БОЛЬШОЙ КОМОК ГРЯЗИ (BIG BALL OF MUD).** Выполняя обзор имеющихся систем, мы обнаруживаем, что существуют части системы, в которых модели перемешаны, а границы стерты. Нарисуйте границу вокруг такой смеси и обозначьте ее как БОЛЬШОЙ КОМОК ГРЯЗИ. Не пытайтесь применять изощренное моделирование внутри этого КОНТЕКСТА. Учтите, что такие системы часто проникают в другие КОНТЕКСТЫ.

Интеграция с *контекстом идентификации и доступа* позволяет *контексту сотрудничества* и *контексту управления гибким проектированием* избежать ОТ-

ДЕЛЬНОГО СУЩЕСТВОВАНИЯ с точки зрения безопасности и полномочий. Действительно, шаблон ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ может применяться в рамках КОНТЕКСТА конкретной системы, но в то же время его можно использовать в каждом случае отдельно. Например, одна команда может отказаться использовать централизованную систему безопасности и решить интегрироваться с другими стандартными системами корпорации.

Команды могут вступать в сотрудничество, играя роли ЗАКАЗЧИК–ПОСТАВЩИК. Руководство компании SaaSOvation не может позволить одной команде вынудить остальных стать КОНФОРМИСТАМИ. Это не значит, что роль КОНФОРМИСТА всегда является отрицательной. Просто шаблон ЗАКАЗЧИК–ПОСТАВЩИК требует, чтобы часть ПОСТАВЩИКА обеспечивала поддержку ЗАКАЗЧИКА, способствуя укреплению отношений между командами, которое руководство компании SaaSOvation считает необходимым для достижения цели. Разумеется, ЗАКАЗЧИКИ не всегда правы, поэтому должен существовать определенный компромисс. Но в целом это полезное организационное отношение, которое команды должны поддерживать.

Для поддержки интеграционных связей между командами можно использовать СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ и ОБЩЕДОСТУПНЫЙ ЯЗЫК. Кроме того, команды могут создать ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ, хотя это может показаться странным. Это не противоречие, даже несмотря на то что между их ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ установлены открытые стандарты. Они могут по-прежнему извлекать выгоду из изолированной трансляции, используя ее основные принципы при взаимодействии с нижестоящими КОНТЕКСТАМИ, но при этом по сравнению с БОЛЬШИМ КОМКОМ ГРЯЗИ уровень сложности снижается. Уровни трансляции становятся простыми и элегантными.

В последующих КАРТАХ КОНТЕКСТОВ используются следующие аббревиатуры, обозначающие шаблоны на каждом из концов отношения.

- ACL (ANTICORRUPTION LAYER — ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ)
- OHS (OPEN HOST SERVICE — СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ)
- PL (PUBLISHED LANGUAGE — ОБЩЕДОСТУПНЫЙ ЯЗЫК)

Анализируя следующий пример КАРТ КОНТЕКСТОВ и сопровождающих их текстов, полезно вернуться к главе 2, “Предметные области, подобласти и ограниченные контексты”. Кроме того, полезно вспомнить три примера ОГРАНИЧЕННЫХ КОНТЕКСТОВ. Поскольку эти диаграммы относятся к высокому уровню, их можно включать в КАРТЫ каждого КОНТЕКСТА, хотя здесь они не повторяются.

Карты трех контекстов

Вернемся к нашей команде, на ошибках которой мы пытаемся научить вас принципам DDD. . .

Когда команда CollabOvation поняла, что создала петлю, ее члены погрузились в изучение книги [Эванс] в поисках выхода. Они не только поняли огромную важность стратегических проектных шаблонов, но и открыли для себя практический инструмент под названием КАРТЫ КОНТЕКСТОВ. Кроме того, они нашли в Интернете статью [Brandolini], описывающую этот метод. Поскольку в статье было сказано, что они должны нарисовать карту существующей территории, команда решила, в первую очередь, сделать именно это. Результаты показаны на рис. 3.2.

Первая КАРТА, созданная командой, отражает первичное представление о существовании ОГРАНИЧЕННОГО КОНТЕКСТА под названием *контекст сотрудничества*. Довольно странная граница этого контекста свидетельствует о том, что команда подозревает о существовании второго КОНТЕКСТА, но не понимает, где проходит четкая граница, отделяющая его от СМЫСЛОВОГО ЯДРА.

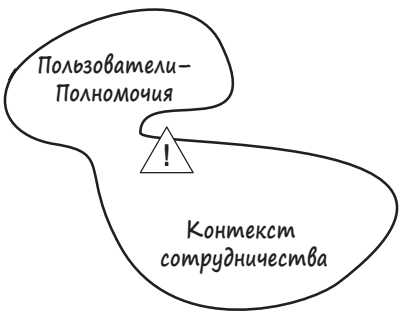


Рис. 3.2. Петля внутри *контекста сотрудничества* вызвана нежелательными концепциями, выявленными в этой *карте*. Предупреждающий знак отмечает засоренную область

Узкий проход в верхней части карты позволяет внешним концепциям мигрировать вперед и назад практически без цензуры. Именно об этом предупреждает знак опасности. Это не значит, что границы КОНТЕКСТА должны быть полностью непроницаемыми. Просто, как и в случае настоящих границ государства, команда хочет, чтобы *контекст сотрудничества* владел полной информацией о том, что пересекает его границы и с какой целью. В противном случае территория становится проходным двором для неизвестных и нежелательных визитеров. Нежелательные визитеры обычно вызывают путаницу и ошибки в моделях. Разработчики моделей должны быть приветливыми и даже гостеприимными, но при условии сохранения порядка и гармонии. Любые внешние концепции, пересекающие границы, должны продемонстрировать право на это, даже приобретая характеристики, совместимые с внутренними свойствами территории.

Этот анализ привел к более глубокому пониманию не только текущего состояния модели, но и направления, в котором должен развиваться проект. Поняв, что такие концепции, как безопасность, пользователи и полномочия, не принадлежат *контексту сотрудничества*, команда смогла принять правильное решение. Команда должна была отделить эти концепции от СМЫСЛОВОГО ЯДРА и позволить их включение только при благоприятных условиях.

Это жизненно важное свойство проекта DDD. Для того чтобы все модели оставались ясными, должны соблюдаться правила ЯЗЫКА каждого ОГРАНИЧЕННОГО КОНТЕКСТА. Лингвистическая сегрегация и строгое выполнение ее правил помогут каждой команде, вовлеченной в проект, сосредоточить внимание на своем ОГРАНИЧЕННОМ КОНТЕКСТЕ и своих задачах.

Анализ ПОДОБЛАСТИ, или пространства задач, привел команду к диаграмме, показанной на рис. 3.3. Оказалось, что ОГРАНИЧЕННЫЙ КОНТЕКСТ состоит из ПОДОБЛАСТЕЙ. Поскольку желательно, чтобы между ПОДОБЛАСТЯМИ и ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ существовало однозначное соответствие, этот анализ продемонстрировал необходимость разделить один ОГРАНИЧЕННЫЙ КОНТЕКСТ на два.

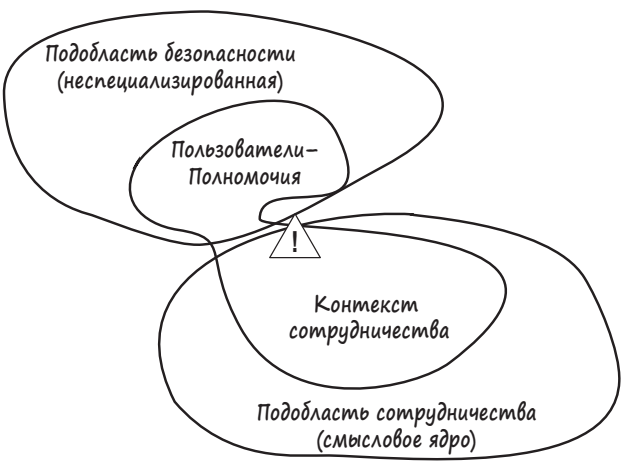


Рис. 3.3. Анализ ПОДОБЛАСТИ позволил обнаружить две новые подобласти: *смысловое ядро сотрудничества* и *неспециализированную подобласть безопасности*

Анализ ПОДОБЛАСТИ и ее границ привел команду к принятию решений. Когда пользователи системы CollabOvation взаимодействуют с доступными функциями, они делают это как УЧАСТНИКИ (PARTICIPANTS), АВТОРЫ (AUTHORS), МОДЕРАТОРЫ (MODERATORS) и т.д. Разнообразие других возможностей, касающихся разделения контекста, мы обсудим позднее, а пока целесообразно разделить созданный контекст. Зная это, можно провести четкие границы, очерчивающие высокоуровневую КАРТУ КОНТЕКСТОВ, как показано на рис. 3.4. Для рефакторинга контекста команда использовала шаблон **ВЫДЕЛЕННОЕ ЯДРО (SEGREGATED CORE)** [Эванс]. Распознанные границы каждого КОНТЕКСТА обозначаются пиктограммами или визуальными знаками. Выделение таких же фигур во всех диаграммах может облегчить процесс познания предметной области.

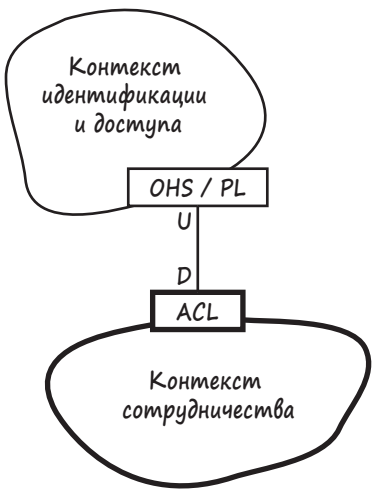


Рис. 3.4. Исходное СМЫСЛОВОЕ ЯДРО обозначено сплошной линией и точками интеграции. В данном случае система IdOvation служит НЕСПЕЦИАЛИЗИРОВАННОЙ ПОДОБЛАСТЬЮ по отношению к нижестоящей системе CollabOvation

КАРТЫ КОНТЕКСТОВ не появляются все сразу, как это может показаться, хотя после завершения анализа их нетрудно воспроизвести. Размышления и обсуждения помогают уточнить КАРТУ путем быстрых итераций. Некоторые уточнения могут выразиться в виде точек интегрирования, описывающих отношения между КОНТЕКСТАМИ.

Первые две карты отображают результаты, достигнутые после применения стратегического проектирования. После того как исходный проект CollabOvation был готов, команда должна была выполнить рефакторинг системы идентификации

и доступа. В итоге она создала КАРТУ КОНТЕКСТОВ, изображенную на рис. 3.4. Команда нарисовала только **СМЫСЛОВОЕ ЯДРО**, которым является *контекст сотрудничества*, а также новую НЕСПЕЦИАЛИЗИРОВАННУЮ ПОДОБЛАСТЬ *контекста идентификации и доступа*. Члены команды не стали рисовать другие будущие модели, такие как *контекст управления гибким проектированием*. Это не слишком помогло бы делу. Им было необходимо исправить лишь то, что уже существовало. Трансформации служебных систем, которые еще только будут созданы, делать не требовалось, поэтому эту работу команда отложила на будущее.

Заметки на доске

- Можете ли вы, размышляя о своем ограниченном контексте, идентифицировать концепции, которые ему не принадлежат? Если да, нарисуйте новую КАРТУ КОНТЕКСТОВ, на которой показаны желательные КОНТЕКСТЫ и отношения между ними.
- Какие из девяти организационных и интеграционных отношений DDD вы бы выбрали и почему?

Когда был начат следующий проект, в котором использовалась система ProjectOvation, настал момент использовать существующую КАРТУ вместе с новым СМЫСЛОВЫМ ЯДРОМ — *контекстом управления гибким проектированием*. Результаты изображены на рис. 3.5. Эти результаты не совпали с ожидаемыми, хотя их еще даже не программировали. Детали внутри нового КОНТЕКСТА были не до конца понятны, но это понимание должно было появиться в ходе обсуждений. Применение высокоуровневого стратегического проектирования на столь ранней стадии должно было помочь всем командам выяснить свои зоны ответственности. Поскольку эти три высокоуровневые КАРТЫ представляют собой всего лишь уточнение предыдущих, мы сосредоточим внимание именно на них. В этом месте на первый план выходит система SaaSOvation. Компания назначила руководителем нового проекта опытного сотрудника. Имея на вооружении три КОНТЕКСТА и план дальнейшей работы, руководство решило направить на разработку нового СМЫСЛОВОГО ЯДРА лучших проектировщиков.

Некоторые существенные моменты сегрегации уже хорошо понятны. Аналогично *контексту сотрудничества*, в котором пользователи системы ProjectOvation создают продукцию, планируют выпуски, разрабатывают календарный план для спринтов и работают над выбором задач, в *контексте управления гибким проектированием* пользователи делают все это как ВЛАДЕЛЬЦЫ ПРОДУКТА (PRODUCT OWNERS) и ЧЛЕНЫ КОМАНДЫ (TEAM MEMBERS). *Контекст идентификации и доступа* выделен из СМЫСЛОВОГО ЯДРА. То же самое касается *контекста сотрудничества*. Теперь это СЛУЖЕБНАЯ ПОДОБЛАСТЬ. Любое ее использование в новой модели будет защищено границами и трансляциями в концепции СМЫСЛОВОГО ЯДРА.

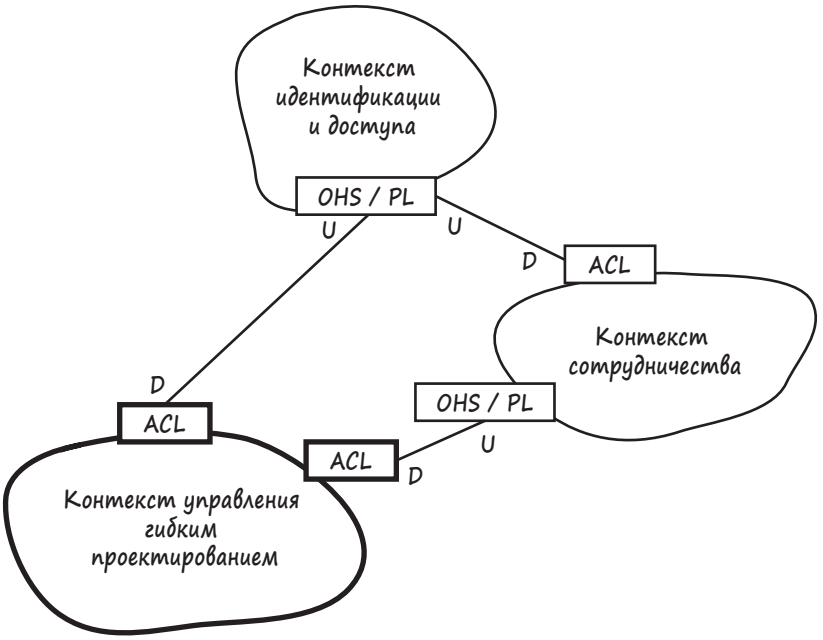


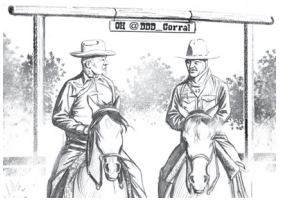
Рис. 3.5. Текущее СМЫСЛОВОЕ ЯДРО обозначено сплошной линией и точками интеграции. СЛУЖЕБНАЯ ПОДОБЛАСТЬ CollabOvation и НЕСПЕЦИАЛИЗИРОВАННАЯ ПОДОБЛАСТЬ IdOvation являются нижестоящими

Рассмотрим более мелкие детали этих диаграмм. Это не архитектурные диаграммы системы. Если бы это было так, то новое СМЫСЛОВОЕ ЯДРО — *контекст управления гибким проектированием* — должно было бы располагаться вверху или в центре диаграммы. В данном случае оно расположено внизу. Эта, возможно, тонкая особенность свидетельствует о том, что модель ядра является нижестоящей по отношению к другим.

Этот нюанс служит также визуальным сигналом. Вышестоящие модели оказывают влияние на нижестоящие, как города, стоящие выше по реке, положительно или отрицательно влияют на города, расположенные ниже по течению. Представьте себе загрязнения, выбрасываемые в реку крупным городом. Эти загрязнения могут мало влиять на сам город, но города, расположенные ниже по течению, могут столкнуться с серьезными последствиями. Вертикальная близость моделей, расположенных на диаграмме, помогает идентифицировать влияние вышестоящих моделей на нижестоящие. Метки U и D возле линий, соединяющих связанные модели, явно указывают на это. Эти метки делают вертикальное позиционирование каждого КОНТЕКСТА менее важным, хотя визуально они выглядят привлекательно.

Ковбойская логика

LB: Если очень хочешь пить, пристраивайся на водопое выше стада по течению.



Контекст идентификации и доступа — это следующий вышестоящий контекст. Он влияет на *контекст сотрудничества* и *контекст управления гибким проектированием*. Наш *контекст сотрудничества* также является вышестоящим по отношению к *контексту управления гибким проектированием*, потому что гибкие модели зависят от модели сотрудничества и служб. Как указано в главе 2, система ProjectOvation работает автономно, поскольку это практично. Операции должны быть максимально независимыми от доступности окружающих систем. Это не значит, что автономные системы должны работать совершенно независимо от вышестоящих моделей. Проектирование должно существенно ограничивать прямые зависимости в реальном времени. Несмотря на свою автономность, *контекст управления гибким проектированием* остается нижестоящим по отношению к остальным.

Установка приложения с автономными службами не означает, что базы данных из вышестоящих КОНТЕКСТОВ просто реплицируются в зависимый КОНТЕКСТ. Репликация вынудила бы локальную систему принять на себя нежелательные обязательства. Это потребовало бы создания ОБЩЕГО ЯДРА (SHARED KERNEL), которое уничтожило бы автономность.

Обратите внимание на узлы, расположенные на вышестоящей стороне каждого соединения на последней КАРТЕ. Оба соединения помечены аббревиатурой OHS/PL, идентифицирующей СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ и ОБЩЕДОСТУПНЫЙ ЯЗЫК. Все три узла соединения на нижестоящей стороне помечены аббревиатурой ACL, означающей ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ. Техническая реализация этих узлов описана в главе, посвященной **ИНТЕГРАЦИИ ОГРАНИЧЕННЫХ КОНТЕКСТОВ (13)**. Коротко говоря, эти шаблоны интегрирования обладают следующими характеристиками.

- **СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ.** Этот шаблон можно реализовать в виде REST-ресурсов, с которыми взаимодействует клиент ОГРАНИЧЕННЫХ КОНТЕКСТОВ. Обычно мы представляем СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ как удаленный вызов процедуры (Remote Procedure Call — RPC) из интерфейса прикладного программирования, но ее можно реализовать и с помощью обмена сообщениями.
- **ОБЩЕДОСТУПНЫЙ ЯЗЫК.** Его можно реализовать несколькими способами, но часто его представляют в виде XML-схемы. Если выразить

ОБЩЕДОСТУПНЫЙ ЯЗЫК с помощью REST-служб, то он будет выражаться через представления концепций предметной области. Эти представления могут включать в себя, например, форматы XML и JSON. Его можно также выразить как буферы протокола Google (Google Protocol Buffers). Если вы публикуете интерфейсы веб-пользователей, то они могут также включать в себя HTML-представления. Одно из преимуществ использования архитектурного стиля REST состоит в том, что клиент может указать предпочтительный ОБЩЕДОСТУПНЫЙ ЯЗЫК, а ресурсы сгенерируют представления в соответствии с требуемым типом контента. Стиль REST также позволяет создавать гипермедийные представления, использующие архитектуру HATEOAS. Гипермедиа делает ОБЩЕДОСТУПНЫЙ ЯЗЫК очень динамичным и интерактивным, позволяя клиентам перемещаться по связанным ресурсам. ЯЗЫК может стать общедоступным благодаря использованию стандартных и/или специальных типов медиа. ОБЩЕДОСТУПНЫЙ ЯЗЫК используется также в **СОБЫТИЙНО-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЕ (EVENT-DRIVEN ARCHITECTURE) (4)**, в которой **СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN EVENTS) (8)** передаются как сообщения всем заинтересованным подписчикам.

- **ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ. СЛУЖБУ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN SERVICE) (7)** для каждого типа ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ можно определить в нижестоящем контексте. Кроме того, ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ можно разместить за интерфейсом **ХРАНИЛИЩА (REPOSITORY) (12)**. Если используется архитектура REST, то реализация клиента СЛУЖБЫ ПРЕДМЕТНОЙ ОБЛАСТИ обращается к СЛУЖБЕ С ОТКРЫТЫМ ПРОТОКОЛОМ. Ответы сервера образуют представления на ОБЩЕДОСТУПНОМ ЯЗЫКЕ. Нижестоящий ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ транслирует представления в объекты предметной области в своем локальном КОНТЕКСТЕ. Здесь, например *контекст сотрудничества* запрашивает у *контекста идентификации и доступа* роль ПОЛЬЗОВАТЕЛЬ–МОДЕРАТОР. Он может получить требуемый ресурс в формате XML или JSON, а затем транслировать его в экземпляр класса `Moderator`, представляющий собой ОБЪЕКТ-ЗНАЧЕНИЕ. Новый экземпляр класса `Moderator` отражает некую концепцию в терминах нижестоящей, а не вышестоящей модели.

Выбранные шаблоны носят общий характер. Ограничение выбора помогает сохранять под контролем объем интеграции, обсуждаемой в этой книге. Как мы увидим, даже среди этих немногих избранных шаблонов существует большое разнообразие способов их применения.

Остается вопрос “Это все, что есть для создания КАРТЫ КОНТЕКСТОВ?” Возможно. Представление высокого уровня обеспечивает хороший объем знаний о

проекте в целом. Однако нас может заинтересовать, куда идут связи и что означают именованные отношения на каждом КОНТЕКСТЕ. Любопытство членов команды стимулирует нас к детализации. Когда мы увеличиваем масштаб, то несколько размытое изображение трех шаблонов интеграции становится более ясным.

Давайте ненадолго вернемся назад. Поскольку *контекст сотрудничества* был первым СМЫСЛОВЫМ ЯДРОМ, давайте всмотримся в него. Сначала мы введем метод увеличения масштаба на примере более простой интеграции, а затем перейдем к более сложным вариантам.

Контекст сотрудничества

Вернемся к нашей команде сотрудничества. . .

Контекст сотрудничества был первой моделью и системой — первым ОСНОВНЫМ ЯДРОМ, — и его принципы работы в данный момент хорошо понятны. Интеграция, используемая здесь, более простая, но менее устойчивая с точки зрения надежности и автономии. Создание масштабируемой КАРТЫ КОНТЕКСТОВ оказалось сравнительно простой задачей.

В качестве клиента REST-служб, опубликованных *контекстом идентификации и доступа*, *контекст сотрудничества* использует для получения ресурсов традиционные вызовы удаленных процедур. Этот КОНТЕКСТ не записывает данные, полученные от *контекста идентификации и доступа*, на которые он может впоследствии сослаться для локального повторного использования. Вместо этого он обращается к удаленной системе, чтобы запрашивать информацию каждый раз, когда в ней возникает потребность. Этот *контекст*, очевидно, очень зависит от удаленных служб и не автономен. Система SaaSOvation на данный момент готова к работе. Интеграция с НЕСПЕЦИАЛИЗИРОВАННОЙ ПОДОБЛАСТЬЮ была абсолютно неожиданна. Для того чтобы выполнить ее напряженный график поставки, команда не могла тратить время на создание более тщательно продуманного автономного проекта. В то же время нельзя было отказаться от принципов простоты проекта. После развертывания системы ProjectOvation и попытки работать автономно подобные методы могут использоваться и для разработки системы CollabOvation.

Пограничные объекты на масштабируемой КАРТЕ, представленной на рис. 3.6, запрашивают ресурс синхронно. Когда представление удаленной модели получено, пограничные объекты извлекают интересующее их содержание и транслируют его, создавая надлежащий экземпляр ОБЪЕКТА-ЗНАЧЕНИЯ. На рис. 3.7 показана КАРТА ТРАНСЛЯЦИИ, предназначенная для того, чтобы превратить представление в ОБЪЕКТ-ЗНАЧЕНИЕ. Здесь объект класса `User`, играющий роль (`Role`) МОДЕРАТОРА в *контексте идентификации и доступа*, транслируется в ОБЪЕКТ-ЗНАЧЕНИЕ `Moderator` в *контексте сотрудничества*.

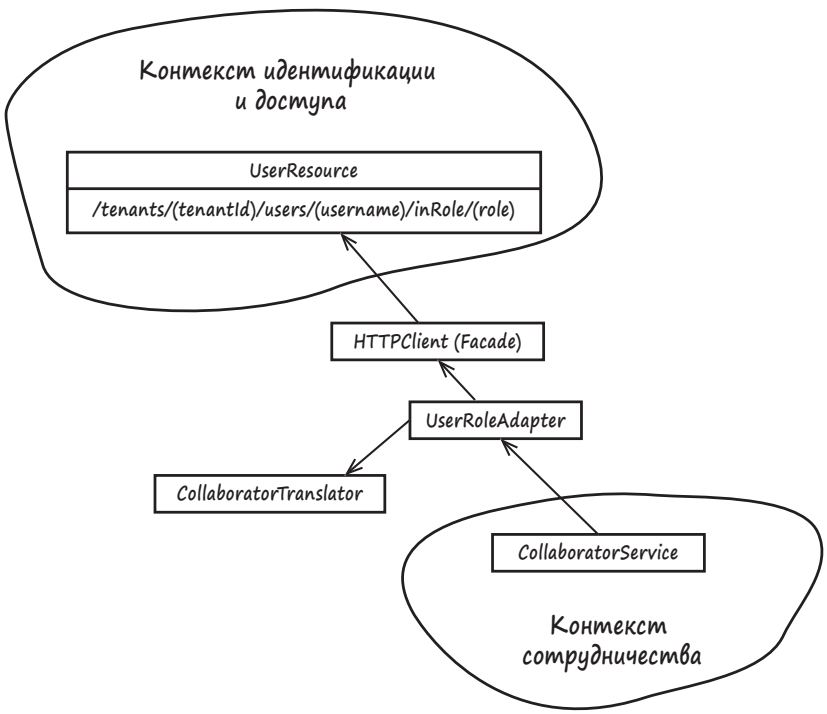


Рис. 3.6. Увеличение масштаба ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ и СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ, участвующих в интеграции между *контекстом сотрудничества* и *контекстом идентификации и доступа*

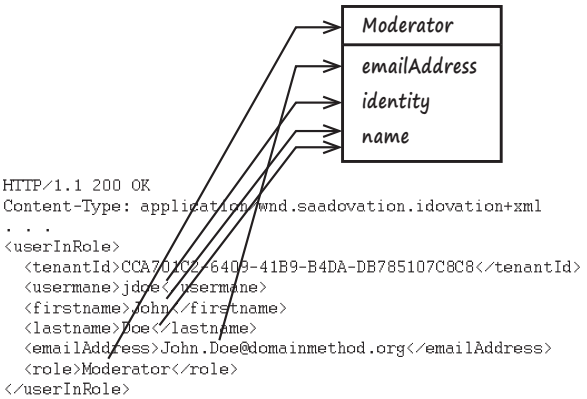


Рис. 3.7. Логическая КАРТА ТРАНСЛЯЦИИ, демонстрирующая, как состояние представления (в данном случае — XML) отображается в ОБЪЕКТ-ЗНАЧЕНИЕ

Заметки на доске

Создайте КАРТУ ТРАНСЛЯЦИИ одного из аспектов интеграции, обнаруженных в ограниченном контексте вашего проекта.

> Что случится, если трансляции окажутся слишком сложными и потребуют копирования и синхронизации слишком большого объема данных, так что транслированный объект будет похож на объект из другой модели? Возможно, это свидетельствует о том, что вы используете слишком много объектов из внешнего ОГРАНИЧЕННОГО КОНТЕКСТА, адаптируете слишком много информации из этой модели и создаете путаницу в своей модели.

К сожалению, если произошел отказ при синхронном запросе из-за того, что удаленная система недоступна, выполнение всех локальных процессов должно прекратиться. Пользователю сообщат о проблеме и попросят попробовать еще раз позже.

Системная интеграция обычно полагается на удаленные вызовы процедур. На высоком уровне удаленные вызовы процедур похожи на обычные вызовы процедур в языках программирования. Библиотеки и инструменты делают вызов привлекательным и простым в использовании. Однако в отличие от вызова процедуры, которая находится в вашем собственном пространстве процесса, удаленный вызов имеет более высокую вероятность снижения производительности или прямого отказа. Загрузка сетевой и удаленной системы может задержать завершение

удаленного вызова процедуры. Если целевая система удаленного вызова процедуры будет недоступна, то запрос пользователя к вашей системе не завершится успешно.

В то время как использование REST-ресурсов на самом деле не сводится к удаленным вызовам процедур, у него все же есть аналогичные характеристики. Несмотря на то, что отказ всей системы — относительно редкое событие, это потенциально раздражающее ограничение. Команда надеется изменить к лучшему эту ситуацию как можно скорее.

Контекст управления гибким проектированием

Поскольку *контекст управления гибким проектированием* — новое СМЫСЛОВОЕ ЯДРО, он заслуживает особого внимания. Увеличим его масштаб и рассмотрим его связи с другими моделями.

Для того чтобы повысить степень автономности по сравнению с удаленными вызовами процедур, команда, разрабатывающая *контекст управления гибким проектированием*, должна осторожно ограничить его использование. По этой причине была принята внеполосная, или асинхронная, обработка событий.

Большая степень автономии может быть достигнута, если в нашей локальной системе уже существует зависимое состояние. Можно представить себе кеш всех зависимых объектов, но при использовании DDD это обычно не так. Вместо этого мы создаем локальные объекты предметной области, транслированные из внешней модели, поддерживая только минимальное количество состояний, необходимых в локальной модели. Для того чтобы вывести состояние на первый план, мы, возможно, должны выполнить удаленные вызовы процедур или подобные запросы REST-ресурсов. Однако любой необходимой синхронизации с изменениями удаленной модели часто можно достичь посредством уведомлений, публикуемых удаленными системами в виде сообщений. Уведомления могли бы быть отправлены по сервисной шине или через очередь сообщений или опубликованы через систему REST.

> **Стремитесь к минимализму**
>
> Синхронизированное состояние является ограниченным, минимальным атрибутом удаленных моделей, необходимым для локальных моделей. Вопрос не только в том, чтобы ограничить наши потребности в синхронизации данных, но и в правильном моделировании концепций.

Целесообразно ограничить использование удаленного состояния, даже рассматривая возможность проектирования элементов локальной модели. Например, объекты классов `ProductOwner` и `TeamMember` в реальности не должны отражать объекты классов `UserOwner` и `UserMember`, потому что они получают

так много характеристик удаленного объекта User, что невольно возникает гибридизация.

Интеграция с контекстом идентификации и доступа

Анализируя увеличенную КАРТУ, изображенную на рис. 3.8, мы видим, что ресурсы URI обеспечивают уведомление о значительных СОБЫТИЯХ ПРЕДМЕТНОЙ ОБЛАСТИ, происходящих в *контексте идентификации и доступа*. Они доступны благодаря экземпляру класса NotificationResource, который публикует ресурс RESTful. Ресурсы уведомлений — это группы публикуемых СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ. Каждое опубликованное СОБЫТИЕ всегда доступно для получения в порядке поступления, но каждый клиент несет ответственность за предотвращение получения дубликатов.

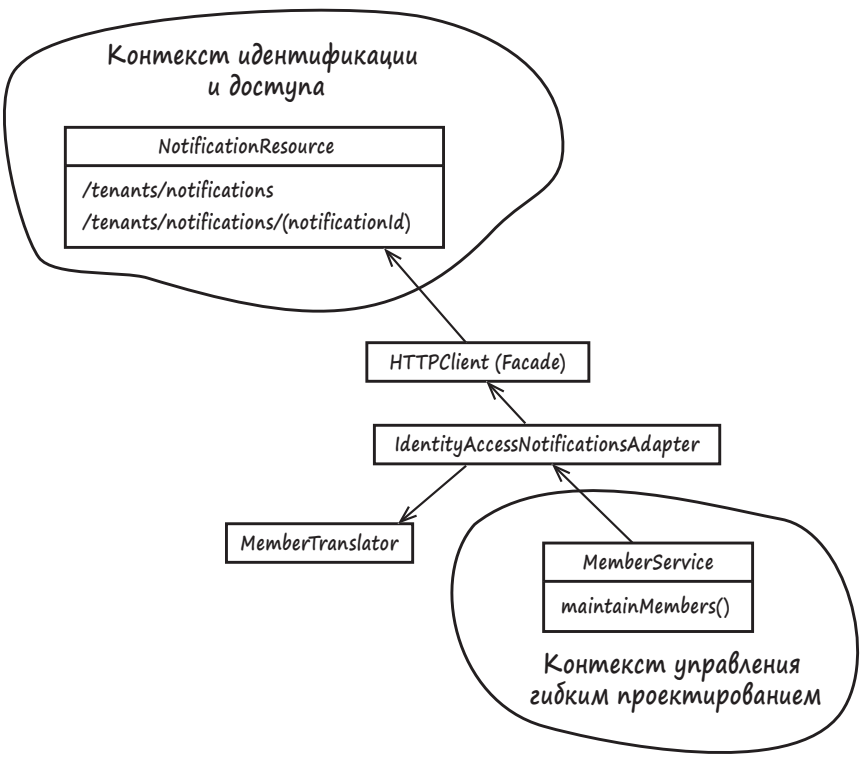


Рис. 3.8. Увеличение масштаба ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ и СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ, участвующих в интеграции между *контекстом сотрудничества* и *контекстом идентификации и доступа*

Тип пользовательского носителя означает, что нам могут понадобиться два ресурса.

```
application/vnd.saasovation.idovation+json
//iam/notifications
//iam/notifications/{notificationId}
```

Адрес URI первого ресурса позволяет клиентам получать (точнее с помощью запроса HTTP GET) журнал текущих уведомлений (фиксированный набор индивидуальных уведомлений). В типе документированного пользовательского носителя

```
application/vnd.saasovation.idovation+json
```

Адрес URI ресурса считается стабильным, потому что никогда не изменяется. Независимо от того, из чего состоит текущий журнал уведомления, этот URI обеспечивает его. Текущий журнал — это ряд новых событий, которые произошли в модели идентификации и доступа. Второй адрес URI ресурса позволяет клиентам получить и перемещаться по цепочке всех предыдущих основанных на событии уведомлений, которые были заархивированы. Зачем нужен текущий журнал и разные заархивированные журналы уведомлений? Подробности, касающиеся уведомлений, передаваемых по каналам, изложены в главах 8 и 13.

На самом деле команда ProjectOvation не обязана использовать архитектуру REST во всех случаях. Например, в настоящий момент она ведет переговоры с командой CollabOvation об использовании альтернативной инфраструктуры передачи сообщений. Кроме того, рассматривается возможность использования платформы RabbitMQ. Тем не менее пока их механизмы интеграции с *контекстом идентификации и доступа* основаны все же на архитектуре REST.

Теперь отложим в сторону технологические подробности и рассмотрим роль, которую играет каждый объект в крупномасштабной КАРТЕ. Ниже приводится объяснение этапов интеграции, продемонстрированных на рис. 3.9.

- Класс `MemberService` — это СЛУЖБА ПРЕДМЕТНОЙ ОБЛАСТИ, ответственная за передачу экземпляров классов `ProductOwner` и `TeamMember` в локальную модель. Она представляет собой интерфейс базового ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ. В частности, метод `maintainMembers()` периодически используется для проверки новых уведомлений, поступающих от КОНТЕКСТА ИДЕНТИФИКАЦИИ И ДОСТУПА. Этот метод не вызывается обычными клиентами модели. При очередном срабатывании таймера компонент, получивший уведомление, использует объект класса `MemberService`, вызывая метод `maintainMembers()`. На рис. 3.9 получателем уведомлений от таймера служит экземпляр класса `MemberSynchronizer`, делегирующий его обработку службе `MemberService`.
- Служба `MemberService` делегирует уведомление экземпляру класса `IdentityAccessNotificationAdapter`, играющему роль АДАПТЕРА между

СЛУЖБОЙ ПРЕДМЕТНОЙ ОБЛАСТИ и удаленной СЛУЖБОЙ С ОТКРЫТЫМ ПРОТОКОЛОМ. АДАПТЕР действует как клиент удаленной системы. Взаимодействие с удаленным объектом класса `NotificationResource` не показано.

- Как только АДАПТЕР получает ответ от удаленной СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ, он делегирует его экземпляру класса `MemberTranslator` для перевода носителя ОБЩЕДОСТУПНОГО ЯЗЫКА в концепции локальной системы. Если локальный экземпляр класса `Member` уже существует, трансляция обновляет существующий объект предметной области. Этот факт показан как самоделегирование сообщения от объекта класса `MemberService` его внутреннему методу `updateMember()`. Класс `Member` имеет подклассы `ProductOwner` и `TeamMember`, отражающие локальные концепции контекста.

Мы не будем рассматривать технологии или средства интеграции. Вместо этого, четко выделив ОГРАНИЧЕННЫЕ КОНТЕКСТЫ, мы сможем сохранить чистоту каждого КОНТЕКСТА, применяя данные из других КОНТЕКСТОВ для выражения своих концепций.

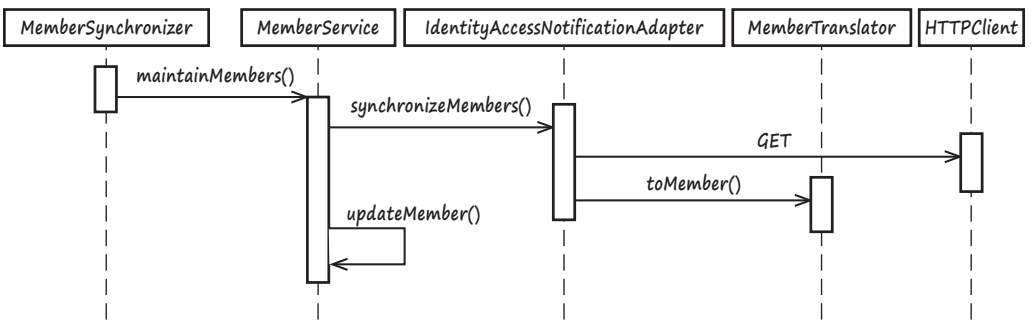


Рис. 3.9. Представление внутреннего функционирования КОНТЕКСТА УПРАВЛЕНИЯ ГИБКИМ ПРОЕКТИРОВАНИЕМ и ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ

Диаграммы и комментарии демонстрируют, как создавать документы, сопровождающие КАРТЫ КОНТЕКСТА. Они не должны быть обширными и в то же время обязаны предоставлять новым членам команды достаточно полную информацию. В общем, документ стоит писать, только если он полезен для команды.

Интеграция с контекстом сотрудничества

Теперь рассмотрим, как *контекст управления гибким проектированием* взаимодействует с *контекстом сотрудничества*. Здесь мы снова будем бороться за автономию, обсуждая интересные вопросы, связанные с обеспечением независимости системы.

Система ProjectOvation получает от системы CollabOvation дополнительные функции. В частности, к ним относятся форумы, посвященные объекту, и общие календари. Пользователи не должны взаимодействовать с системой CollabOvation непосредственно. Система ProjectOvation должна самостоятельно определять, доступны ли те или иные функциональные возможности для конкретного арендатора, и в случае положительного ответа создавать собственный вспомогательный ресурс в системе CollabOvation.

Рассмотрим раздел сценария использования *Создание продукта*.

Предусловие: существует возможность сотрудничества (средство куплено).

1. Пользователь предоставляет описательную информацию о продукте.
2. Пользователь выражает желание обсудить ее с командой.
3. Пользователь посылает запрос на создание указанного продукта.
4. Система создает продукт, а также форум и обсуждение.

Форум и обсуждение должны быть созданы в *контексте сотрудничества* от имени продукта. В противоположность этому вряд ли в *контексте идентификации и доступа* уже предусмотрен агент, определены пользователи, группы и роли и доступны сообщения об этих событиях. В *контексте сотрудничества* эти объекты уже существуют, но в *контексте управления гибким проектированием* требуются объекты, которые еще не существуют и не должны существовать, пока на них не поступит запрос. Это может мешать автономии, потому что при создании удаленного объекта мы зависим от доступности *контекста сотрудничества*. Стремясь к автономии, мы должны решить интересную проблему.

> **Почему *обсуждение* используется в обоих *контекстах***
>
> Возникла интересная ситуация, потому что одно и то же имя, *обсуждение*, появляется в обоих ОГРАНИЧЕННЫХ КОНТЕКСТАХ, имеющих разные типы, разные объекты, а значит, разное состояние и разное поведение.
>
> Обсуждение в *контексте сотрудничества* представляет собой АГРЕГАТ и управляет множеством постов — неявных наследников, которые сами являются АГРЕГАТАМИ. Обсуждение в *контексте управления гибким проектированием* — это ОБЪЕКТ-ЗНАЧЕНИЕ и лишь содержит ссылку на реальное обсуждение с постами во внешнем КОНТЕКСТЕ. Отметим, однако, что в главе 13 команда, выполняющая интеграцию, сталкивается с необходимостью строгой типизации разных видов обсуждений в *контексте управления гибким проектированием*.

Для поддержания согласованности необходимо использовать шаблоны **СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ (8)** и **СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (4)**. Ничто не свидетельствует о том, что только удаленные системы могут получать уведомления, созданные нашей локальной системой. Когда в нашей

модели публикуется СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ProductInitiated, оно обрабатывается нашей системой. Локальный обработчик отправляет запрос на создание удаленных форума и обсуждения. Это можно сделать либо с помощью вызова удаленных процедур, либо путем передачи сообщений, в зависимости от того, какой механизм поддерживает система CollabOvation. Если в данный момент механизм вызова удаленных процедур и системы удаленного сотрудничества недоступны, локальный обработчик будет вынужден просто периодически пытаться выполнить свою работу, пока она не завершится успешно. Если же передача сообщений поддерживается механизмом вызова удаленных процедур, то локальный обработчик пошлет сообщение системе сотрудничества. В свою очередь, система сотрудничества ответит своим сообщением, когда создание ресурса будет завершено. Когда обработчик СОБЫТИЯ в системе ProjectOvation получает это сообщение, он устанавливает ссылку идентификации в объекте класса Product на вновь созданное обсуждение.

Что происходит, если владелец продукта или члены команды пытаются использовать обсуждение до того, как оно появится? Рассматривается ли недоступное обсуждение как ошибка в модели? Сообщает ли в этом случае система о невыполнимом условии? Обратите внимание на то, что подписчики не обязаны платить за использование системы обеспечения сотрудничества. Это одна из нетехнических причин, по которым в проекте учитывается недоступность ресурса. Обойти проблемы итоговой согласованности, конечно, не получится. Это просто еще одно допустимое состояние, которое следует моделировать.

Элегантный способ решения этой задачи заключается в обработке всех возможных сценариев недоступности системы, чтобы они стали явными. Рассмотрим данный шаблон **СТАНДАРТНЫЙ ТИП (STANDARD TYPE)**, реализованный как шаблон **СОСТОЯНИЕ (STATE)** [Гамма и др.] и описанный в главе 6.

```
public enum DiscussionAvailability {
    ADD_ON_NOT_ENABLED, NOT_REQUESTED, REQUESTED, READY;
}

public final class Discussion implements Serializable {
    private DiscussionAvailability availability;
    private DiscussionDescriptor descriptor;
 ...

}

public class Product extends Entity {
 ...

private Discussion discussion;
 ...
}
```

Эта схема позволяет защитить ОБЪЕКТ-ЗНАЧЕНИЕ Discussion от неправильного использования, потому что его защищает СОСТОЯНИЕ, определенное экземпляром класса DiscussionAvailability. Когда кто-нибудь пытается принять участие в обсуждении класса Product, он может безопасно передать свое состояние обсуждения. Если это состояние — не READY, то участник получит одно из трех сообщений.

Для участия в командном сотрудничестве необходимо приобрести надстройку.

Владелец продукта не запрашивал создание обсуждения.

Настройка обсуждения еще не завершена; повторите попытку позднее.

Если состояние экземпляра Discussionavailability равно READY, вы получите возможность полноценного командного участия.

Как следует из первого сообщения о состоянии обсуждения, у команды существует возможность выборочного участия в сотрудничестве, даже если это участие не было оплачено. Это может оказаться правильным маркетинговым ходом, стимулирующим дальнейшие покупки. Кто сможет быстрее уговорить руководство приобрести надстройку, чем тот, кто каждый день получает уведомление о том, что он мог бы использовать ее, но пока не может? Очевидно, что доступность СОСТОЯНИЯ позволяет оценить не только технические преимущества.

В настоящий момент команда не уверена в том, какой на самом деле должна быть интеграция с системой сотрудничества. Для того чтобы понять схему обсуждения ЗАКАЗЧИК-ПОСТАВЩИК, она создала рис. 3.10. *Контекст управления гибким проектированием* может использовать второй ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ для интеграции с *контекстом сотрудничества*. Это похоже на то, как он использует *контекст идентификации и доступа*. На диаграмме показаны основные пограничные объекты, похожие на свои аналоги, используемые при интеграции с системой идентификации и доступа. На самом деле в системе нет ни одного объекта класса CollaborationAdapter. Это просто рамка для объекта, который будет нужен, но пока неизвестен.

Внутри локального КОНТЕКСТА показаны объекты классов Discussion-Service и SchedulingService. Они представляют собой СЛУЖБЫ ПРЕДМЕТНОЙ ОБЛАСТИ, которые можно использовать для управления обсуждениями и календарными записями в системе сотрудничества. Реальные механизмы будут определены в ходе переговоров между командами по шаблону ЗАКАЗЧИК-ПОСТАВЩИК, описанному в главе 13.

Теперь команда может понять часть своей модели. Например, что произойдет, если будет создано обсуждение, а результат будет передан в локальный контекст. Асинхронный компонент — системы вызовов удаленных процедур или обработчик событий — передает методу attachDiscussion() класса Product новый

экземпляр ЗНАЧЕНИЯ Discussion. Все локальные агрегаты с отложенными удаленными ресурсами будут обрабатываться именно таким образом.

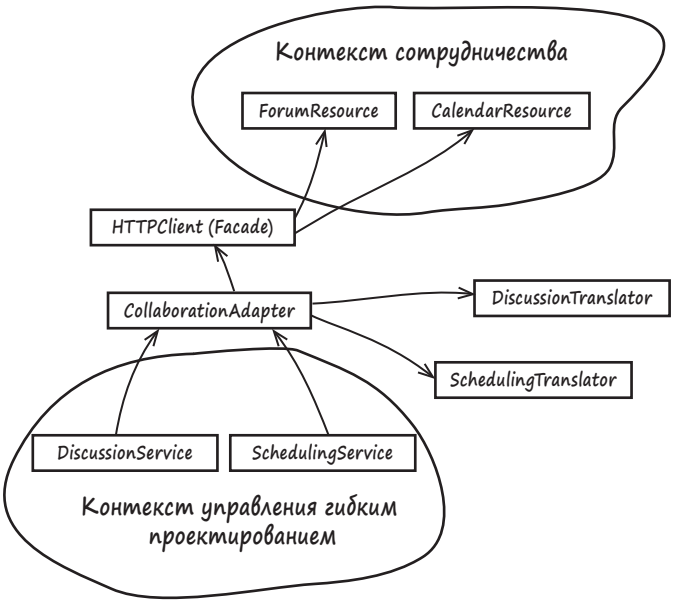


Рис. 3.10. Увеличение масштаба ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ и СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ, участвующих в интеграции между *контекстом сотрудничества* и *контекстом идентификации и доступа* в локальном КОНТЕКСТЕ

Эта проверка выявила интересные детали КАРТ КОНТЕКСТОВ. Однако мы должны ограничиться этим, потому что приближаемся к точке падения эффективности. Вероятно, можно было бы включить в систему **МОДУЛИ (9)**, но им посвящена отдельная глава. Включите в модель все релевантные высокоуровневые элементы, необходимые для обеспечения важных аспектов коммуникации между командами. С другой стороны, остановитесь, когда детали станут слишком подробными.

Нарисуйте КАРТУ КОНТЕКСТОВ, которую можно было бы повесить на стене. Ее можно загрузить в командную базу знаний, поскольку она касается не только команды. Постоянно обсуждайте проект, стараясь уточнить свою КАРТУ.

Резюме

Мы провели очень продуктивную работу над КАРТОЙ КОНТЕКСТОВ.

- Мы обсудили, что собой представляет КАРТА КОНТЕКСТОВ, чем она может помочь команде и как ее легче создать.
- Мы подробно рассмотрели три ОГРАНИЧЕННЫХ КОНТЕКСТА системы SaaSOvation и их вспомогательных КАРТ КОНТЕКСТОВ.
- Используя карты, мы увеличили масштаб интеграции между всеми КОНТЕКСТАМИ.
- Мы исследовали пограничные объекты, поддерживающие ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ и его взаимодействия.
- Мы увидели, как создать КАРТУ ТРАНСЛЯЦИИ, демонстрирующую локальное отображение между REST-ресурсом и соответствующим объектом в модели предметной области.

Не каждый проект требует такой детализации, как этот. Другие проекты могут потребовать еще большей проработки. Необходимо найти баланс между понятностью и практичностью и не слишком зарываться в детали на каждом уровне. Напоминаю, что мы не собираемся создавать очень подробную КАРТУ проекта. Она нужна лишь для того, чтобы во время обсуждений члены команды могли ссылаться на нее. Если отбросить условности и поставить во главу угла простоту и гибкость, то можно создать полезную КАРТУ КОНТЕКСТОВ, которая поможет нам идти вперед, а не увязнуть в болоте трудностей.