Андрей Николаевич Трушкин

АРХИТЕКТУРА ЦИФРОВОГО МИРА

Андрей Трушкин

Архитектура цифрового мира

«Издательские решения»

Трушкин А. Н.

Архитектура цифрового мира / А. Н. Трушкин — «Издательские решения»,

ISBN 978-5-00-560843-7

Настоящая книга посвящена становлению и совершенствованию цифрового мира, отражает связь развития экономики, разделения и углубления труда и сферы ИТ. Автор дает прямые ответы на ключевые вопросы. Как должна развиваться архитектура цифрового мира? Как должен быть выстроен процесс цифровой трансформации, чтобы развитие организации носило перманентно интенсивный характер? Как значимо повысить эффективность инвестиций в ИТ? Книга будет полезна как ИТ-специалистам, так и руководителям организаций.

ISBN 978-5-00-560843-7

Содержание

Архитектура цифрового мира

Андрей Николаевич Трушкин

«Архитектура – это создание порядка света».
Антонио Гауди

ISBN 978-5-0056-0843-7
Создано в интеллектуальной издательской системе Ridero

Введение

Идеи, положенные в основу настоящей работы, возникли у автора достаточно давно, и связаны они были с его непосредственной деятельностью как архитектора-практика в области информационных технологий. И первый вопрос, возникший у него при начале работы над книгой, звучал крайне просто: «Какое название должна книга получить?»

Первым сформулированным предложением было «ИТ-архитектура в цифровом мире». Читатель может подумать, что мы недалеко ушли от него, приняв то название, которым поименована книга, которую он держит в руках или открыл на экране своего электронного устройства. Смеем возразить, что между этими двумя названиями лежит целая эпоха.

Фактически с самого начала активного внедрения информационных технологий в нашу жизнь, а прошло уже почти 40 лет с момента, как они стали уделом не только ученых, но и стали массово применяться в самых различных областях человеческой деятельности, был выделен в качестве обязательного этап проектирования программного обеспечения, сформировано понятие «ИТ-архитектура» в качестве одного из базовых, а многие квалифицированные сотрудники, задействованные в выполнении проектов, стали называться ИТ-архитекторами. Конечно, ИТ-архитектура далеко не во всем соответствовала афоризму Шеллинга («Архитектура – это музыка, застывшая в камне»), но она предоставляла общий взгляд на создававшиеся информационные системы, позволяла понять как сложность их реализации, так и перспективы, открывавшиеся по итогам внедрения.

Информационные технологии не только активным образом внедрялись во все сферы мировой экономики (сначала стран Западного мира, а потом и бывшего социалистического блока), но и изменяли мир. Безусловно, подобные изменения требовали времени и не всегда соответствовали зачастую завышенным ожиданиям. На рубеже XX—XXI веков ряд консалтинговых компаний представил доклады, которые фиксировали факт отсутствия существенного роста производительности труда в компаниях, активно внедрявших автоматизацию в производственную деятельность, от средней по экономике в общем или конкретной ее отрасли. Например, в октябре 2001-го года был опубликован доклад компании McKinsey «Рост производительности труда в США в 1995—2000», в котором констатировался факт, что на рассмотренном историческом периоде рост производительности труда в компаниях, внедрявших информационные технологии (далее ИТ), не превышал среднего значения по рынку. Исключением являлись случаи, когда ИТ сами по себе являлись новыми средствами производства. На тот момент исследования столь уважаемых консалтинговых компаний заставляли с опаской относиться к внедрению в проверенные рыночные сегменты новых направлений. И тем не менее, изменения продолжались – всё новые и новые компании самых разных областей деятельности активно переходили к автоматизации своих производственных и бизнес-процессов. Наряду с автоматизацией компаний, представляющих традиционные отрасли экономики, в начале XXI века произошел стремительный рост технологических и финтех-компаний (FinTech), создавались социальные платформы, приобретшие международный характер, что вносило коренные изменения в структуру экономики как развитых, так и развивающихся стран. В 2010-е годы изменения приобрели драматический характер.

Чем же обусловлено подобное изменение? В 2001-ом году говорилось об отсутствии значительного влияния ИТ-технологий на производительность труда и изменение структуры экономики. Спустя 20 лет (не такой большой срок по меркам истории человечества, даже если акцентировать внимание на периоде интенсивного развития, обычно отсчитываемого от Славной революции в Англии – 1688 года), когда автор приступил к созданию настоящей книги, рынок цифровых технологий является самым быстро растущим в мире, более того, в нем открываются все новые и новые возможности для инвестирования, участники традиционных

секторов экономики стремительно трансформируются в соответствии с новыми реалиями. Мы видим, что совершенно различные компании декларируют собственную трансформацию в ИТ-компании, предоставляющие финансовые, коммуникационные, логистические и иные услуги. Как правило, сложные процессы развития и внедрения технологий (не только ИТ) описываются так называемой S-кривой. Пример S-кривой приведен на Рисунке 1.

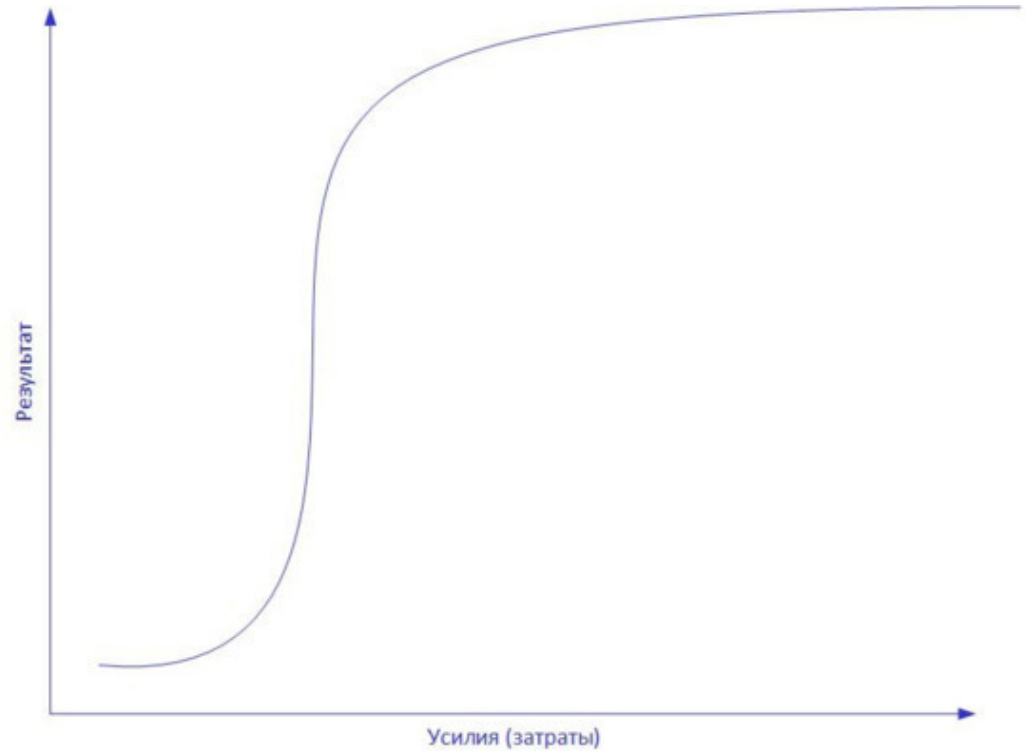

Рисунок 1. Пример S-кривой

Традиционно закон S-кривой отражает изменения ключевых показателей происходящих в сложных системах изменений, при этом количественный характер изменений носит описательный характер вспомогательного свойства. Упор данного закона делается на качественные изменения. В соответствии с законом S-кривой, развитие технологий проходит три ключевые стадии:

• Рабочий подготовительный этап, характеризующийся значительными инвестициями финансовых, временных и трудовых ресурсов с относительно невысокими результатами, часто его так и называют – этапом подготовки. Примером работ, выполняемых на данном этапе, могут служить научно-исследовательские и опытно-конструкторские работы.

• Этап интенсивного внедрения технологий, на котором происходит захват рынка, сопровождаемый резким ростом популярности технологии, при этом принципиально меняется характер отдачи инвестиций, вкладываемых в соответствующее направление, в зависимости от эффективности технологии именно на данном этапе происходит отдача инвестиций и получение прибыли.

• Этап стагнации, при наступлении которого технология уже не дает серьезной отдачи, а потому на смену ей приходят новые направления, в свою очередь характеризующиеся рассматриваемым законом.

Необходимо отметить, что далеко не все технологии и направления развития проходили все указанные этапы (в том числе, последовательно) – многие либо не оправдывали ожиданий,

«сваливаясь» с кривой развития и уходя в небытие, к некоторым возвращались на новом историческом, научно-техническом и технологическом уровне.

На значимых исторических периодах можно увидеть актуальность данного закона для самых различных сфер деятельности, для частных решений и глобальных технологических тенденций. В качестве наглядного примера можно привести энергетический сектор экономики.

Если говорить об энергетике, следует привести в пример уголь, бывший ключевым ресурсом в течение XIX – начала XX века, сменившую его в данном качестве нефть, а также возобновляемые источники энергии (ВИЭ), качественное положение которых на соответствующей S-кривой представлено на Рисунке 2.

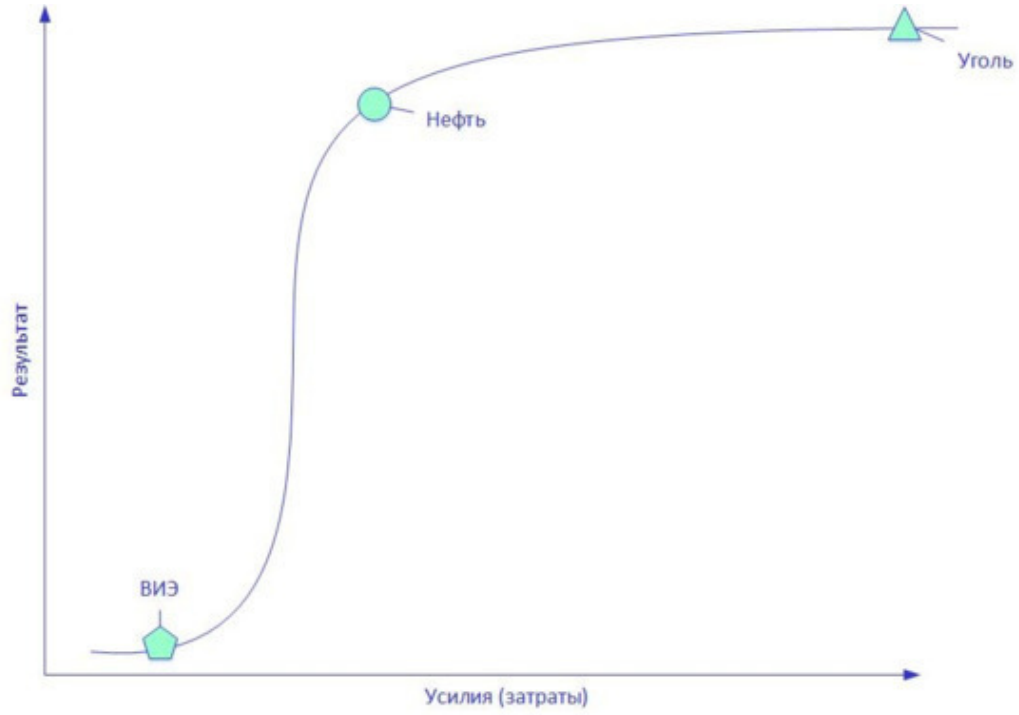

Рисунок 2. S-кривая источников энергии

Угольная генерация прошла этап активного внедрения и рассматривается современными гигантами мировой экономики в качестве во многом устаревшего направления приложения усилий. Соответствующая отрасль не генерирует значимой прибыли, но по-прежнему может использоваться для балансировки энергетических потоков, представляя собой технологически надежное и проверенное временем решение. Нефтяная генерация крайне бурно развивалась в течение XX века, но в последнее время вошла в стадию, когда многие ключевые игроки соответствующего рынка направляют свои усилия в смежные области, диверсифицируя направления деятельности. Капитальные инвестиции в данном секторе уже не показывают того бурного роста, который был характерным для ушедшего века. Что же касается ВИЭ, то данному направлению еще предстоит доказать свою актуальность и перспективность, перейдя в фазу роста. Речь при этом идет как о направлении в целом, так и о конкретных технологиях, входящих на сегодня в сам термин ВИЭ, носящий комплексный характер.

Если вернуться к ИТ, то можно утверждать, что для данного направления также характерно развитие, описываемое S-кривой. На основании приведенных выше исследований, а также актуальных тенденций рынка, выскажем мнение, что для сегмента ИТ характерна точка перелома – переход от подготовительного этапа к этапу интенсивного внедрения технологий,

при этом указанный переход еще нельзя считать свершившимся. Текущее положение ИТ показано на Рисунке 3.

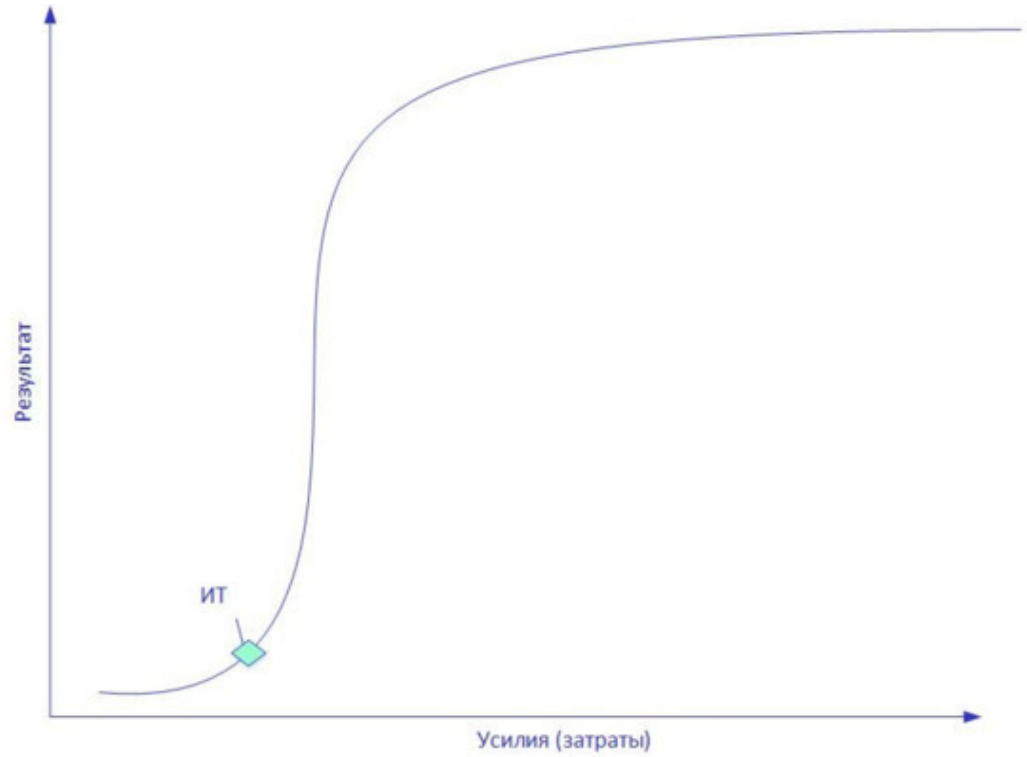

Рисунок 3. S-кривая ИТ

Упомянутые выше доклады рубежа веков вполне корректно описывали подготовительный этап – компании самых различных секторов экономики активно инвестировали в ИТ, внедряли соответствующие технологии в повседневную работу, но эффективность их производственных и бизнес-процессов не показывала качественных изменений по сравнению с партнерами и конкурентами, пренебрегавшими указанными инвестициями. Отметим, что непосредственно сами консалтинговые агентства на тот момент не использовали собранную фактическую базу для построения долгосрочных прогнозов относительно глобальных перспектив всего сектора ИТ. Время подтвердило правомерность такой позиции.

Спустя двадцать лет сложно представить жизнь без ИТ. И речь в данном случае пойдет не о легкомысленных занятиях, которые предпочитают приводить в пример некоторые консерваторы. Проведем мысленный исторический эксперимент: представим себе пандемию коронавирусной инфекции SARS-CoV-2 и связанный с ее начальным периодом практически общемировой локдаун в 2001-ом году с соответствующим уровнем развития ИТ. Нетрудно, проведя соответствующую экстраполяцию, прийти к выводу, что если сегодня спад экономики той или иной страны, вызванный противоэпидемическими мероприятиями, является предметом дискуссии, то в предложенном эксперименте мировая экономика попросту не пережила бы соответствующих мер. Возможности удаленной работы, новый технологический уровень медицинской, финансовой, промышленной сфер, принципиально иной коммуникационный уровень качественно изменили и возможности ответа даже на столь серьезные вызовы, одним из которых и стала пандемия. На фоне приведенного примера изменения, произошедшие в конкретных сферах общественной жизни, можно считать частными:

• Переход финансового сектора к формату дистанционного обслуживания в качестве приоритетного, что привело к качественно новому предоставлению услуг и нагрузочной способности.

• Моделирование сельскохозяйственной деятельности, позволяющее на порядки повысить эффективность расходования удобрений и аналогично снизить число людей, занятых в соответствующем секторе экономики.

• Переход к безлюдному производству с новым уровнем эффективности.

• Новые коммуникационные возможности, включающие, например, вытеснение традиционных голосовых звонков звонками через мессенджеры.

Приведенный перечень специализированных отраслевых изменений является далеко не полным – его исчерпывающее перечисление не является задачей настоящей работы.

С учетом вышеизложенного мы можем с полным правом сказать, что за последние 20 лет мир стал действительно цифровым.

Но если мир стал цифровым, то следует вернуться к вопросу, которым был открыт текущий раздел: почему было принято решение отказаться в названии книги от приставки «ИТ» применительно к архитектуре. Как уже отмечалось выше, понятие ИТ-архитектуры было введено практически одновременно с началом активного внедрения ИТ в различные сегменты человеческой деятельности, введен в качестве обязательного при создании новых информационных систем этап проектирования, сформулированы требования к роли ИТ-архитектора. Всё это произошло еще в первой половине 80-х годов XX века. Но с тех пор происходили колоссальные изменения не только в части влияния ИТ на мир, рассмотренного выше, но и влияния мира на ИТ. Концептуально это взаимовлияние представлено на Рисунке 4.

Рисунок 4. Взаимовлияние ИТ и мира

ИТ активным образом внедрялись в самые различные отрасли экономики, качественно изменяя их, но и сами изменялись в рамках этого внедрения. В условиях трансформации лидеров экономических секторов в ИТ-компании неверно продолжать утверждать, что ИТ является уделом избранных – по сути, все сотрудники этих компаний начинают работать в ИТ, в том самом цифровом мире, который уже построен, очертания которого сформированы на наших глазах. Таким образом, нам кажется в корне неверным продолжать употреблять термины «ИТ-архитектура», «архитектура информационных систем» и т. д. Правильным будет говорить об архитектуре обсуждаемого цифрового мира. Именно данный термин и будет использоваться в книге в дальнейшем. Более того, архитектура становится образом мыслей, к ней может применяться термин **mindset**, имеющий зачастую более широкое значение. Указанный аспект мы подробно рассмотрим в соответствующей главе.

Данное терминологическое уточнение исключительно важно в части подчеркивания направленности настоящей книги. Возвращаясь к обсуждению проблематики S-кривой ИТ, необходимо задать себе следующий вопрос: «Если мир уже можно назвать цифровым, если неправильно говорить об ИТ, предполагая соответствующее направление уделом избранных,

то почему при обсуждении S-кривой мы говорим лишь о точке перелома, но не о полноценной фазе интенсивного внедрения?»

Говоря о структуре и закономерностях самой S-кривой, можно отметить, что те секторы (технологии, направления), которые успешно преодолели (преодолевают) фазу интенсивного развития, отличались на соответствующем этапе исключительно высокой с точки зрения актуального временного периода наукоемкостью. Формировались самостоятельные научные дисциплины, посвященные соответствующим направлениям, создавалось единое концептуальное и смысловое пространство. К сожалению, на сегодня нельзя констатировать подобное формирование в случае ИТ и архитектуры. Само понимание архитектуры зачастую носит принципиально разный характер, определяемый спецификой деятельности конкретного специалиста. Иногда, говоря об архитектуре, подразумевают сетевую схему информационной системы. В некоторых случаях речь идет о наборе технологий, применяемых в конкретном решении. Иногда об инфраструктуре, обеспечивающей функционирование организации.

В настоящей книге не рассматриваются вопросы составления сетевых схем, организации межсетевого экранирования, инфраструктурного обеспечения подобных процессов. Автор оставляет данные вопросы специалистам, ни в коем случае не отрицая, тем не менее, их важность. Указанный круг проблем напрямую связан с аспектом информационной безопасности, который не может игнорироваться с учетом той структуры современного мира, которая была выстроена в последние десятилетия. Тем не менее, это может считаться лишь частным вопросом архитектуры. Мы рассмотрим вопросы связи архитектуры и информационной безопасности в соответствующем разделе.

Подробное рассмотрение проблем использования тех или иных технологий является исключительно обширным и глубоким вопросом. В качестве примера на Рисунке 5 приведен список технологий, которые могут использоваться в рамках комплексного проекта по автоматизации. Список не является полным или исчерпывающим и приводится исключительно в качестве наглядного примера сложности вопроса. Современные решения, позволяющие автоматизировать целые отрасли экономики, не ограничиваются одной из технологий – они требуют достаточно широкого технологического спектра, при этом правила и границы применимости соответствующих частных технологических компонентов могут отличаться от случая к случаю. Отметим также, что технологический набор, представленный на Рисунке 5, не является рекламой конкретных продуктов, а лишь иллюстрирует сложности состава современных решений, актуальных для цифрового мира.

Подробное рассмотрение структуры и особенностей каждой технологии, применимой для использования в комплексном проекте автоматизации, выйдет за пределы любой книги. Отметим, что для ИТ характерна исключительно высокая скорость развития, поэтому подробное рассмотрение каждой технологии сопряжено с неизбежным устареванием подобного исследования. Автор не претендует на пальму первенства в части проведения подобных исследований, оставляя ее представителям сообществ, непосредственно занимающихся развитием соответствующих технологий.

Рисунок 5. Пример списка технологий для комплексного проекта автоматизации

Вместе с тем, необходимо отметить, что комплексные исследования, ставящие в центре рассмотрения основную проблематику ИТ в целом, рассмотрение ИТ в качестве научной дисциплины, на данный момент крайне немногочисленны. Данный факт не позволяет утверждать, что для ИТ уже актуальна фаза интенсивного развития. Большинство работ посвящены частным вопросам, как в случае с обеспечением информационной безопасности, либо направлены на обеспечение глубокого понимания конкретных технологий. Автор стремился восполнить данный пробел, направляя свои усилия именно на интенсификацию развития ИТ. Соответствующим образом и построена настоящая книга. Частные случаи и конкретные технологические решения будут упоминаться в качестве примеров, иллюстрирующих комплексное рассмотрение.

Тенденции и ключевые направления развития

Ключевые тренды развития

Переход от стадии подготовки (Рисунок 3) к стадии интенсивного развития ИТ сопряжен с формированием ключевых трендов, оказывающих влияние на развитие как отрасли, так и всего цифрового мира. Точка перелома сопряжена именно с качественным формированием соответствующих тенденций, при этом характеризуется недостаточным следованием им: не все участники цифрового мира принимают важность той или иной тенденции, многие из которых незаслуженно игнорируются, внедряются хаотично, зачастую и вовсе объявляются не значимыми с точки зрения архитектуры. По ходу настоящей главы в соответствующих разделах будут рассмотрены ключевые тенденции, предпосылки их возникновения и оформления, а также последующие перспективы. Настоящий раздел посвящен общему рассмотрению данных тенденций.

Во введении говорилось о том, что в течение последних двадцати лет рынок ИТ пережил качественные изменения, при этом подчеркивалась значимость с точки зрения изменений 2010-х годов. Что же стало драйвером произошедших изменений?

В 2001-ом году из знаковых для ИТ событий следует отметить не только выход упоминавшегося выше доклада компании McKinsey, не только кризис доткомов (Dot-com bubble), завершение которого традиционно датируется именно 2001-ым годом, показавший избыточность инвестиций в акции Интернет-компаний и перегрев соответствующего рынка, но и публикацию манифеста гибкой разработки программного обеспечения (Agile Manifesto), называемого также манифестом Agile. Связь приведенных событий не следует недооценивать и сводить лишь к хронологической близости. Кризис доткомов породил неуверенность в эффективности внедрения ИТ и перспективах соответствующего рынка. Данная неуверенность стала основой вопросов, отмеченных консалтинговыми компаниями, в том числе McKinsey. Манифест Agile же был попыткой ответа со стороны профессионального сообщества на возникавшие вопросы и претензии к отрасли, ответа, предлагающего варианты повышения производительности труда и эффективности ИТ в целом. Отметим важность данного манифеста не столько в методологическом смысле, а именно таким образом иногда пытаются сузить его обсуждение, сколько в мировоззренческом. Вопросы методологии в разработке программного обеспечения стали подниматься намного раньше, уходя корнями еще в тот период, когда ИТ были уделом в большей степени ученых, не находя широкого применения в различных отраслях человеческой деятельности. Рассмотрение применимости различных методологий для современной на тот момент разработки программного обеспечения стало одной из составных частей дипломной работы автора настоящей книги. При этом манифест Agile декларировал мировоззренческое изменение ИТ.

В задачи настоящей книги не входит обсуждение пунктов манифеста. Отметим следующие значимые эффекты следования ему. Создание новых информационных систем, внедрение ИТ в производственные и бизнес-процессы предлагалось вести с учетом максимальной автономности команд профессионалов, максимально быстрой готовности минимального жизнеспособного продукта (MVP), постоянного циклического улучшения собственной работы. Манифест стал декларацией открытости ИТ миру, готовности к изменениям. От него можно отсчитывать старт взаимовлияния ИТ и мира, построение цифрового мира.

Изначально новые подходы к ИТ были востребованы не крупными корпорациями с их устоявшимися подходами, характеризующимися развитой бюрократической составляющей, а стартапами, нацеленными на получение быстрых результатов в новых областях ИТ. Резуль-

таты работы ряда стартапов стали качественным изменением взгляда на ИТ. На основе некоторых стартапов, успешно применявших гибкие методологии разработки и изменявших свое мировоззрение, были созданы глобальные технологические компании, деятельность которых на сегодняшний день вышла за рамки строго рынка ИТ: в их обсуждении участвуют ученые, политики, бизнесмены, военные, философы, социологи и люди самых разных профессий. В качестве примеров таковых компаний можно отметить Meta Platforms (ранее Facebook) и Google. Было бы значительным упрощением утверждать, что подобное качественное изменение было достигнуто исключительно следованием манифесту Agile, но последний запустил ряд тенденций, которые и являются предметом рассмотрения настоящей главы.

Внимание стартапов к работе по новым принципам вывело на новый уровень интерес к решениям с открытым исходным кодом и распространяемым на свободной основе. По мере роста стартапов, обусловленного этим повышения интереса к гибким методологиям, решения, бывшие ранее узкоспециализированными, вышли на новые рынки и стали использоваться в глобальном масштабе. Сами стартапы, становившиеся технологическими компаниями мирового масштаба, сохраняли исходную свободную культуру, в рамках которой создававшиеся технологические решения становились достоянием сообщества разработчиков. В качестве примера можно привести Apache Cassandra – одно из наиболее популярных решений в части распределенных систем управления базами данных, применяемое при создании высоконадежных хранилищ данных. Проект был разработан в компании Facebook, а в 2009 году передан фонду Apache Software Foundation. На сегодня данное решение (с открытым исходным кодом) применяется в крупнейших компаниях различных сфер деятельности по всему миру (IBM, Huawei, Netflix, Apple, Spotify и др.). Отметим, что все технологии, представленные ранее на Рисунке 5, основаны на открытом исходном коде.

Изменения в подходе к использованию технологий требуют и изменения подхода к архитектуре, самих архитектурных концепций. Если при использовании программного обеспечения, предоставляемого поставщиком в «закрытом» (vendor-lock) формате, возможным было учитывать наличие экспертизы компании-поставщика, то в случае использования программного обеспечения, основанного на открытом исходном коде, предполагающего внимание к деталям внедрения, необходим совершенно иной уровень компетенции архитектора, глубины проработки архитектуры, изменение самой культуры последней. Поэтому использование решений, основанных на открытом исходном коде, и стало первой ключевой тенденцией развития архитектуры.

Активное использование гибких методологий стартапами, формирование на их основе ряда технологических гигантов, создание последними на базе решений с открытым исходным кодом технологических продуктов, используемых для автоматизации гигантских информационных систем мирового масштаба, заставляют учитывать в качестве одного из ключевых трендов развития архитектуры фактор распределенности. Распределенность может рассматриваться в двух смысловых плоскостях. Во-первых, речь идет о территориально распределенной структуре команд разработки (в данном случае речь идет не только о программистах, но о всех специалистах, участвующих в процессе создания решений). Во-вторых, современные ИТ-решения должны активным образом масштабироваться, не снижать качество предоставления услуг при высокой нагрузке, допускать выполнение в различных часовых поясах, странах и континентах. Оба указанных аспекта оказывают ключевое влияние на архитектуру, ее развитие, на процессы проектирования новых решений. В качестве примера можно привести социальные платформы, например, Facebook. Технологические решения Meta Platforms (ранее Facebook) обрабатывают запросы на предоставление услуг миллиардов пользователей со всех континентов, при этом качество предоставления этих услуг регулярно находится на высшем уровне. По мере становления цифрового мира данные требования стали всеобщими. Если рассмотреть временной период 10 лет от характеризующегося значительным количеством знаковых

для ИТ событий 2001-го года, то можно отметить, что, например, для финансовых организаций ключевым каналом взаимодействия с клиентами было отделение, а нагрузочная способность внедрявшихся ИТ-решений рассчитывалась на основании числа отделений, среднего потока клиентов через них, при этом делалась поправка на распределение отделений по часовым поясам. Спустя еще 10 лет – в настоящее время – вводятся понятия Bank as a Service (BaaS), Bank as a Platform (BaaP), которые сами по себе свидетельствуют о коренном изменении самой модели предоставления финансовых услуг – все ключевые каналы на сегодня дистанционные, при этом характер продвижения продуктов посредством данных каналов во многом соответствует модели социальных платформ. Приведенный пример характерен не только для финансового сектора экономики, но и практически для всех остальных, что и позволило еще во введении говорить о цифровом мире. При этом над созданием соответствующих решений трудятся команды, распределенные по всему миру. Работая в соответствии с гибкими методологиями, они создают актуальные продукты в итеративном и инкрементном режиме, регулярно представляя результаты своего труда клиентам и партнерам. Таким образом, архитектура должна обеспечивать возможность эффективной работы соответствующих команд, возможность достижения ими результата независимо друг от друга, при этом создаваемые решения должны обеспечивать нагрузочную способность качественно нового порядка, географически распределенный режим функционирования. Распределенность можно определить второй ключевой тенденцией развития архитектуры.

В современных организациях определяются сотни и тысячи бизнес-процессов, эффективное исполнение которых формирует пульс жизни самой организации. Традиционным архитектурным решением для автоматизации выполнения бизнес-процессов было включение в структуру ИТ-ландшафта организации компонента, обычно именуемого Business Process Manager (BPM), который позволяет последовательно выполнять шаги всех бизнес-процессов, требующих автоматизации. Отметим, что современные бизнес-процессы состоят из сотен автоматизируемых действий, при этом в организациях, следующих в своей работе гибким методологиям, в бизнес-процессы постоянно вносятся изменения, применение которых в режиме промышленной эксплуатации востребовано в реальном времени. Традиционное централизованное исполнение бизнес-процессов общим компонентом BPM ограничивает возможности независимой работы команд разработчиков, а также вызывает вопросы к собственной нагрузочной способности. Современная архитектура должна обеспечивать гибкую автоматизацию бизнес-процессов, при которой управление бизнес-процессом децентрализовано, отдельные компоненты бизнес-процесса автоматизируются независимыми командами разработчиков, немедленно внедряющими вносимые ими изменения в промышленную эксплуатацию. Соответственно, определяется третья ключевая тенденция развития архитектуры, связанная с автоматизацией бизнес-процессов.

Одним из краеугольных камней современного цифрового мира являются данные. При этом за последние годы (можно сказать, что даже за последние 10 лет) существенным образом изменился сам подход к данным. Во многом основой изменений стала деятельность технологических гигантов, выросших из вчерашних стартапов. Еще десять лет назад многие компании традиционных секторов экономики в течение длительного периода времени накапливали данные о своих клиентах, продуктах, технологических процессах структурированным образом, предполагавшим сбор информации из анкет, данных органов государственной власти, кредитных бюро, истории взаимодействия с организацией и/или ее несколькими партнерами (с которыми были заключены договоры, предусматривавшие возможность передачи соответствующих данных). Затем на основе собранных данных строились аналитические витрины, позволявшие не только формировать обязательную отчетность, но и определять показатели эффективности и их достижение, планировать разработку новых продуктов организации и т. д. При этом построение аналитических витрин занимало продолжительное время,

сами же витрины и выводы на их основе носили статистический характер. В последние 10 лет под влиянием технологических гигантов в части работы с данными произошли существенные изменения. Созданы социальные платформы международного характера, пользователи которых предоставляют сведения о себе, носящие как структурированный, так и неструктурированный (информация о хобби, фотографии, видео и т.д.) характер. В результате автоматизации промышленности собирается колоссальное количество данных о состоянии оборудования, его составных частей, необходимости ремонта, параметрах помещений, где располагается оборудование и т. д. Такое количество данных оказалось фактически невозможно обработать за приемлемый промежуток времени – поломки оборудования происходили за более короткий промежуток времени, нежели требовался для расчета вероятности поломки.

На основании изменившегося характера и количества данных требовались и новые подходы к их обработке. Пионерами здесь также стали технологические гиганты. Новые средства автоматизации работы с данными позволяли обрабатывать качественно новые объемы информации, носящей как структурированный, так и неструктурированный характер, в режиме, приближенном к реальному времени. Более того, результаты обработки зачастую могут носить не статистический, а таргетированный характер. Примерами такой обработки являются лента новостей в социальных сетях и контекстная реклама. Новые возможности, открывавшиеся в результате сбора данных и их обработки современными технологическими средствами, привлекли внимание и представителей традиционных отраслей человеческой деятельности. Сегодня уже не является удивительным, например, формирование комбинированных финансовых продуктов банковскими организациями и их таргетированное предложение клиентам, при этом основанием для формирования и предложения являются предпочтения клиента, представленные им в социальных сетях. Таким образом, данные, их обработка, анализ результатов последней являются исключительно важными аспектами проектирования информационных систем нового поколения во всех областях цифрового мира. Соответственно, следует выделить четвертую ключевую тенденцию развития архитектуры – данные. Отметим также, что большинство современных технологических средств, обеспечивающих качественно новый уровень работы с данными, основаны на принципах открытого кода. Таким образом, можно отметить синергию ключевых тенденций развития архитектуры.

Сбор данных и аналитика, осуществляемая на их основе, дают основание проводить экстраполяцию соответствующей тенденции на принятие решений, выполняемое в автоматическом режиме. Логическим продолжением соответствующих тенденций становится внедрение в различные области человеческой деятельности технологий искусственного интеллекта, позволяющего принимать решения, осуществлять действия, в том числе требующие творческой составляющей и традиционно считающиеся прерогативой человека. Автоматизация соответствующих областей также стала реальностью. Отметим, что на сегодня применение искусственного интеллекта ограничивается решением узкоспециализированных задач: выявление мошенничества, диагностика заболеваний, формирование стратегии интеллектуальных игр (шахматы, го), принятие решений в области рисков, промышленная роботизация. При этом комплексные интеллектуальные технологические решения, объединяющие различные отрасли и сферы деятельности, на сегодня недостаточно развиты, невзирая на имеющийся спрос. Таким образом, следует выделить еще одну (пятую) ключевую тенденцию развития архитектуры, заключающуюся в опоре на искусственный интеллект, позволяющую давать ответ на спрос в части соответствующих решений. Внедрение искусственного интеллекта может кратно (а возможно, и на порядки) повысить эффективность и качество производственных и бизнес-процессов.

Выделенные пять тенденций являются ключевыми современными тенденциями в развитии архитектуры вследствие кратного увеличения качества внедрения ИТ, повышения эффективности соответствующих внедрений и их последующего влияния на сферы, где эти внед-

рения осуществляются. Отметим также, что данные тенденции не следует рассматривать изолированно друг от друга, по ходу настоящего раздела уже приводились примеры их взаимовлияния. В рамках следующих разделов выявленные тенденции будут рассматриваться более подробно, их синергия также будет подробно рассмотрена.

Необходимо отметить, что многие современные тенденции в архитектуре, считающиеся на сегодняшний день уже стандартом де-факто, такие как микросервисная архитектура, событийно-ориентированная архитектура, потоковая передача данных, машинное обучение, продуктовый подход и ряд других, были сформулированы в качестве ответов на те вызовы, которые предъявляются в рамках отмеченных в настоящем разделе тенденций развития. Данная связь также будет проиллюстрирована по ходу дальнейшего изложения.

Отдельно будут рассмотрены вопросы эволюционного и революционного развития архитектуры. Надо отметить, что вопросы соотношения экстенсивного и интенсивного развития характерны для многих отраслей человеческой деятельности. Не избежали их ИТ и цифровой мир. Рассмотрению вопроса соотношения интенсивного и экстенсивного развития в архитектуре цифрового мира будет посвящен следующий параграф.

Эволюционное и революционное развитие архитектуры

Вопросы развития любого направления всегда затрагивают аспекты интенсивного и экстенсивного развития. Первое обычно сравнивают с революцией, второе с эволюцией. Нельзя избежать рассмотрения соответствующих аспектов и в случае архитектуры.

Следует отметить, что сравнение интенсивного и экстенсивного развития носит качественный и обобщающий характер. Если же говорить о частностях, углубляться в детали, то возможны ошибки. Одна и та же тенденция в развитии, следование ей может на определенном этапе носить интенсивный (революционный) характер, с течением времени же интенсивность снижается, развитие осуществляется строго экстенсивно (эволюционно), в дальнейшем переходя к застою и неся в себе риски деградации. В то же время, строго интенсивное развитие в силу своего революционного характера использует в качестве ресурсного обеспечения мощности своей базовой технологии. Таким образом, может произойти разрушение как конкретной технологии, так и всего сектора экономики, при этом высвобождающиеся мощности становятся ресурсным обеспечением революционного рывка. Например, на базе потери интереса к большим машинам, затраты на содержание которых зачастую превышали прибыли от автоматизации, произошел стремительный рост рынка персональных компьютеров и соответствующего программного обеспечения, что в свою очередь привлекло потребителей и последующие инвестиции в данное направление. Не всякое разрушение в таком случае может создать достаточное ресурсное обеспечение революционного перехода, что чревато разрушением с последующей деградацией всего сектора экономики, технологического направления.

Рассмотрим примеры эволюционного и революционного развития архитектуры и выделим их основные характеристики. В качестве наглядного примера рассмотрения возьмем такую ключевую тенденцию развития архитектуры, как распределенность.

Как уже отмечалось, распределенность характеризуется в двух различных смысловых плоскостях: команды разработчиков, создающие конечную ценность для заказчиков и партнеров, распределены географически и работают автономно друг от друга, сами же создаваемые технологические решения функционируют в распределенной конфигурации, сохраняя высокое качество услуг.

Если взять за отправную точку рассмотрения уже упоминавшийся столь значимый для ИТ 2001-й год, можно отметить, что на тот момент ИТ-решения для большинства направлений человеческой деятельности имели «монолитный» характер, представляли собой целостную единицу развертывания (либо весьма ограниченный набор таких единиц). Создание и развитие таких решений велось обширной командой развития, включавшей представителей разных направлений деятельности.

Возьмем в качестве наглядного примера финансовую сферу. Основой финансовой сферы являлась (и является) учетная платформа (в России традиционное название – автоматизированная банковская система, АБС). Эта система (при исходном проектировании) выполняла следующие функции:

- Учет финансовых/хозяйственных операций;
- Ведение счетов;
- Ведение плана счетов;
- Бухгалтерский документооборот;
- Валютный учет;
- Многофилиальный учет;
- Налоговый учет;
- Формирование учетной документации для проведения проверок;

• Формирование оперативной и аналитической бухгалтерской отчетности (операционный день и т. п.).

По мере автоматизации производственных и бизнес-процессов финансовой организации в состав учетной платформы включались исходно несвойственные ей функции:

• Ведение данных клиентов;

• Управление бизнес-процессами;

• Автоматизированные рабочие места сотрудников, ведущих обслуживание клиентов;

• Продуктовые операции (не только учетного характера, но и, например, продуктовые конвейеры);

• И т. д.

При этом учетная платформа представляла собой «закрытое» (vendor-lock) программное обеспечение, основанное на «закрытом» же технологическом стеке, предоставляемом независимыми поставщиками, требовавшая значительных инфраструктурных мощностей для обеспечения корректного функционирования.

Если подходить к указанному примеру (как частному случаю) с точки зрения эволюционного и революционного развития, то следует отметить, что создание первых учетных платформ было развитием революционным (сам факт создания систем и автоматизации соответствующей области деятельности), сменившимся эволюцией (монотонное наращивание целевого функционала учетной платформы, например, основанное на появлении новых регуляторных требований), также чреватое застоем и деградацией (дополнение системы несвойственным ей с точки зрения потребностей бизнеса функционалом).

Пример указанного функционала монолитной системы в формате учетной платформы приведен на Рисунке 6.

На Рисунке 6 представлено разделение учетной платформы на следующие составные блоки:

• Целевой функционал, который направлен на автоматизацию собственно учетной функции;

• Вмененный функционал, который не связан непосредственно с целями учета;

• Автоматизированные рабочие места, к которым можно отнести целевые (АРМ бухгалтерии) и вмененные (АРМ специалиста по обслуживанию клиентов).

Отметим, что Рисунок 6 не претендует на *детальное* отображение функционала учетных платформ, внедрявшихся и разрабатывавшихся в финансовых учреждениях. Также следует отметить, что далеко не все финансовые организации современности смогли исключить вмененный функционал из используемых ими учетных платформ, что существенно влияет на возможность применения гибких практик разработки при внедрении ИТ в них (финансовых организациях) в негативном ключе.


Учетная платформа (АБС)
АРМ БУХГАЛТЕРА
АРМ СПЕЦИАЛИСТА ПО ОБСЛУЖИВАНИЮ КЛИЕНТОВ
Вмененный функционал учетной платформы
Продуктовые операции
Бизнес-процессы
Клиенты
Дедубликация данных
Базовый функционал учетной платформы
Главная книга Банка
Ведение счетов
Бухгалтерский документооборот (счет-фактуры, мемориальные ордера и пр.)
Оперативная и аналитическая бухгалтерская отчетность (отчетные формы, выписки и пр.)
Валютный учет (переоценка счетов и пр.)
План счетов
Операционный день
Многофилиальный учет
Обязательная отчетность
Банк России
Бюджетная система РФ
ФАС России
ФНС России
Проверки
Налоговый учет (формирование сообщений в налоговые органы и пр.)

Рисунок 6. Пример монолитной системы в формате учетной платформы

Развитие учетных платформ осуществлялось в соответствии с длительными производственными процессами, предполагавшими последовательное прохождение ряда стадий, таких как анализ, проектирование, разработка, тестирование и отладка, при этом релизные циклы решений занимали по нескольку месяцев. Различные компоненты решений оказывали взаимовлияние друг на друга и не могли разрабатываться параллельно, либо же их параллельная разработка была крайне затруднена.

К эволюционному переходу в те годы можно отнести создание новых информационных систем, автоматизировавших часть вмененного функционала учетных платформ, и их точечную интеграцию с последними. Таким образом удавалось несколько упростить функционирование отдельных программных комплексов за счет усложнения интеграционных взаимодействий. Указанные изменения не обеспечивали существенного роста эффективности.

Революционным переходом можно было считать переход финансовых организаций того времени (речь идет о передовых организациях соответствующего сектора) к модели сервис-ориентированной архитектуры (service-oriented architecture, SOA), предполагавшей разнесение различных блоков функционала по отдельным автоматизированным системам, каждая из которых предоставляла доступный контрагентам функциональный набор в формате групп сервисов, при этом интеграционные взаимодействия между системами, как предполагалось, должны были осуществляться унифицированным образом. Пример организации SOA представлен на Рисунке 7.

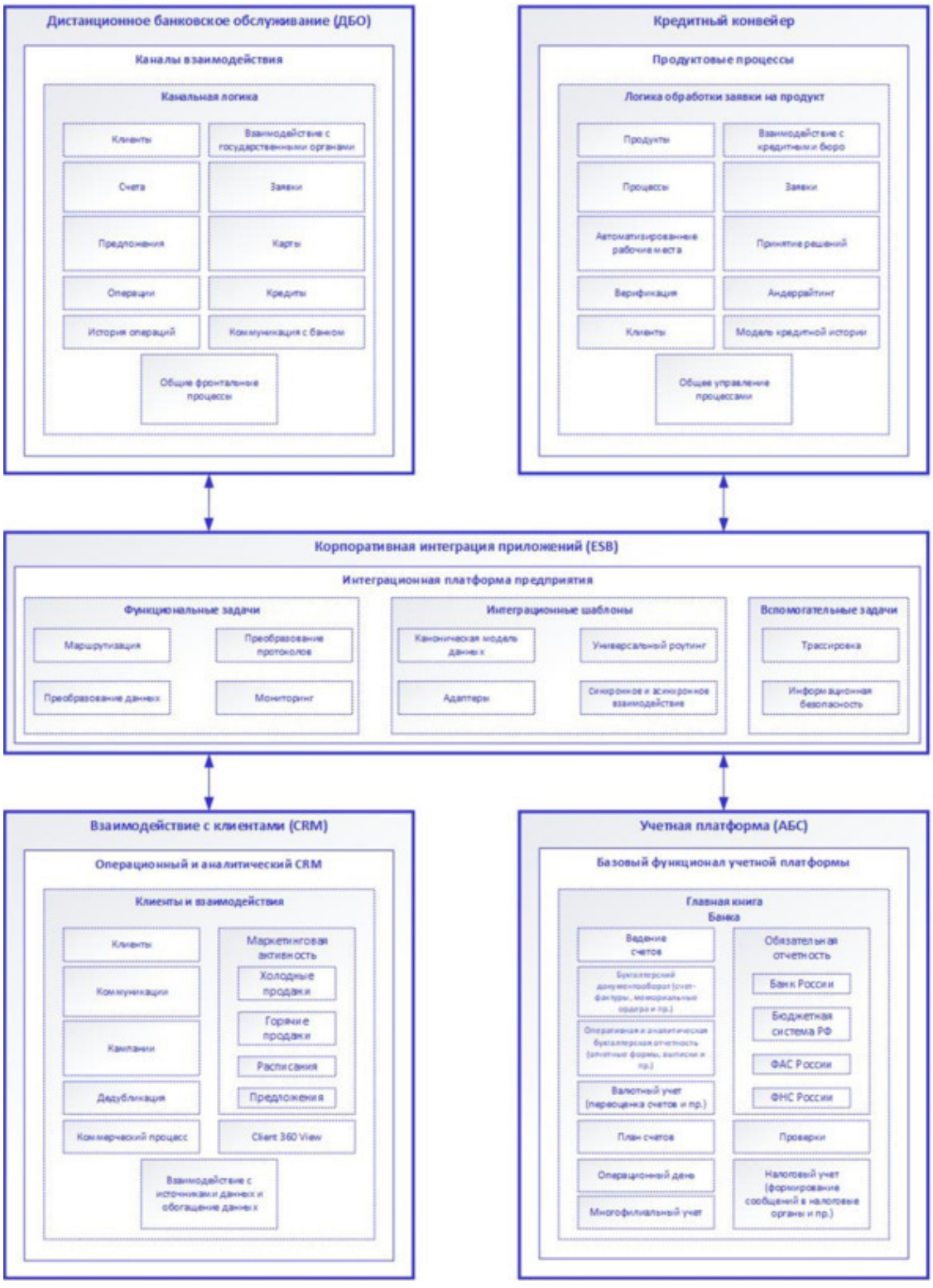

Рисунок 7. Пример организации SOA

Рассмотрим структуру SOA, пример которой представлен на Рисунке 7, более подробно. При внедрении соответствующих архитектурных практик ИТ-ландшафт организации (в нашем примере финансовой) распределялся по функциональному назначению: выполнение учетных функций (учетная платформа), управление взаимодействием с клиентом (система Customer

Relationship Management, CRM), системы дистанционного банковского обслуживания (ДБО), продуктовые конвейеры (на Рисунке 7 в качестве примера приведен кредитный конвейер).

Создатели архитектурных практик SOA стремились выставить акценты при разработке и развитии информационных систем в соответствии с функциональными границами областей, ими автоматизировавшимися. При этом унификация интеграционного взаимодействия осуществлялась посредством реализации шаблона проектирования «Корпоративная сервисная шина» (Enterprise Service Bus, ESB), подробное описание которого приведено в книге Gregor Hohpe, Bobby Woolf «Enterprise Integration Patterns» (2003, ISBN 978—0321200686). Для реализации указанного шаблона применялись, как правило, отдельные информационные системы (было выпущено немало как «закрытых» решений, таких как IBM WebSphere ESB, так и основанных на принципах открытого кода, как Apache ServiceMix). Ключевыми задачами, решавшимися при реализации шаблона ESB, были следующие:

• Обеспечение прозрачной маршрутизации взаимодействий между интегрируемыми информационными системами с избавлением последних от необходимости детального знания о месторасположении контрагентов.

• Преобразование протоколов взаимодействия, что избавляло интегрируемые системы от учета технических особенностей взаимодействия с контрагентами, само же решение ESB, как правило, поддерживало большой набор протоколов, взаимодействуя с системами по базовым протоколам последних.

• Преобразование форматов данных, что обеспечивало возможность каждой интегрируемой системы функционировать в рамках собственной модели данных, при этом список потенциальных контрагентов мог расширяться за счет решения задачи дополнительных преобразований на уровне ESB.

• Мониторинг интеграционных взаимодействий, которые должны были осуществляться строго посредством решения, реализующего шаблон ESB.

Таким образом, решение, реализующее шаблон ESB, оказывалось сквозным в масштабах организации, что привело к дополнению его функционала задачами информационной безопасности, аудита, трассировки и т. д. Кроме того, для реализации непосредственных задач решение реализовывало ряд шаблонов проектирования, представленных как в вышеупомянутом труде («Enterprise Integration Patterns»), так и в одной из основополагающих работ в сфере ИТ: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides «Design Patterns: Elements of Reusable Object-Oriented Software» (1994, ISBN 0-201-63361-2). К числу подобных шаблонов можно отнести каноническую модель данных, адаптер и т. д.

Как уже отмечалось выше, переход к SOA стал революционным в плане развития архитектуры и внедрения ИТ в организациях. В результате разгрузки учетных платформ от несвойственного им функционала передовые в технологическом плане финансовые организации смогли приступить к активной автоматизации своих производственных и бизнес-процессов. Практики SOA стали основными, при этом к началу 2010-х годов ведущие финансовые организации в целом завершили перевод собственной автоматизации в данном ключе. Сформированный ИТ-ландшафт был, безусловно, намного сложнее, чем представленный на Рисунке 7 пример – в крупных финансовых организациях он включал в свой состав сотни и тысячи информационных систем. Можно отметить, что к этому периоду SOA из революционной практики стала эволюционной с ярко выраженной тенденцией к застою и деградации. В чем же выражалась соответствующая тенденция?

Представим себе ситуацию, что в финансовой организации 200 информационных систем, которые выстроены и взаимодействуют между собой в соответствии с принципами SOA. При этом каждая информационная система предоставляет 10 сервисов, посредством которых ее функционал доступен контрагентам. Как результат, на интеграционной шине представлено

2000 сервисов. Каждая из систем развивается независимо в соответствии с планами владельцев систем (предположим, что владельцами систем в организации являются бизнес-подразделения). Соответственно, команда специалистов, отвечающая за развитие ESB решения, должна поддерживать актуальность всех 2000 сервисов, добавлять новые, обеспечивать корректность функционирования всего комплекса. Не будем забывать, что нередким является случай, при котором потребителями одного сервиса являются несколько информационных систем. В таком случае при развитии соответствующего сервиса необходимо также поддерживать механизмы версионности, а также обеспечивать регрессионное тестирование. Емкость любой команды не является бесконечной величиной. Кроме того, важным является тот факт, что ESB не представляет собой конечной значимой бизнес-функциональности, то есть участие членов команды развития ESB решения в создании бизнес-ценности является опосредованным, что ограничивает возможности масштабирования соответствующей команды. Результатом, как правило, являлось то, что скорость внесения изменений на уровне ESB была существенно ниже, нежели в информационных системах, автоматизирующих прикладной функционал. Особенно заметной разница в скорости внесения изменений была при сравнении ESB с системами дистанционного банковского обслуживания (ДБО), где соответствующая скорость традиционно одна из самых высоких в автоматизации финансового сектора. Автор лично был свидетелем того, как уже выполненные доработки уровня ДБО ожидали необходимых интеграционных доработок год и более.

Шаблон ESB и следование ему было не единственной причиной потенциальной деградации. Заказчики и партнеры любой организации (финансовая не является исключением) заинтересованы в *продуктах*, которые она предоставляет. В случае финансовой организации это могут быть вклады, счета, кредиты, банковские гарантии и т. д. В случае следования методикам SOA предоставление продукта клиентам и партнерам предполагало реализацию соответствующего функционала в рамках нескольких информационных систем: канального представления, исполнения процессов, связанных с соответствующим продуктом, принятия решений в части исполняющихся процессов, постановки продукта на бухгалтерский учет и т. д. При этом нельзя забывать, что отдельные системы пусть и предоставляли свой функционал в виде набора сервисов, доступность которых контрагентам обеспечивало ESB решение, сами по себе оставались монолитными комплексами с длительными релизными циклами. Многие из этих систем требовали специфических компетенций для развития. При этом компетенции требовались не только технического, но и методологического характера. В результате выпуск продукта зависел от системы с максимально длительным релизным циклом. На фоне успехов стартапов, ряд которых к началу 2010-х годов уже стали технологическими гигантами, сложившаяся ситуация не могла устраивать представителей традиционных секторов экономики (пример, приведенный в настоящей книге для финансовой организации, является характерным для традиционных секторов экономики в целом). Тенденции, сформировавшиеся после экономического кризиса 2008-го года, также требовали повышения эффективности всех отраслей человеческой деятельности. Возникла необходимость в новом революционном переходе.

Первым этапом нового революционного перехода стала концепция микросервисной архитектуры, в соответствии с которой приложения представлялись в виде набора минимально зависимых (в идеальной ситуации независимых) модулей (микросервисов), каждый из которых представлял собой полноценную единицу развертывания с самостоятельным жизненным циклом. При этом взаимодействие микросервисов предполагалось осуществлять исключительно посредством API, что исключало необходимость владения информацией об особенностях исполнения контрагентов. Данная концепция соответствовала практикам гибкой разработки и DevOps. Пример предлагавшейся концепции представлен на Рисунке 8.

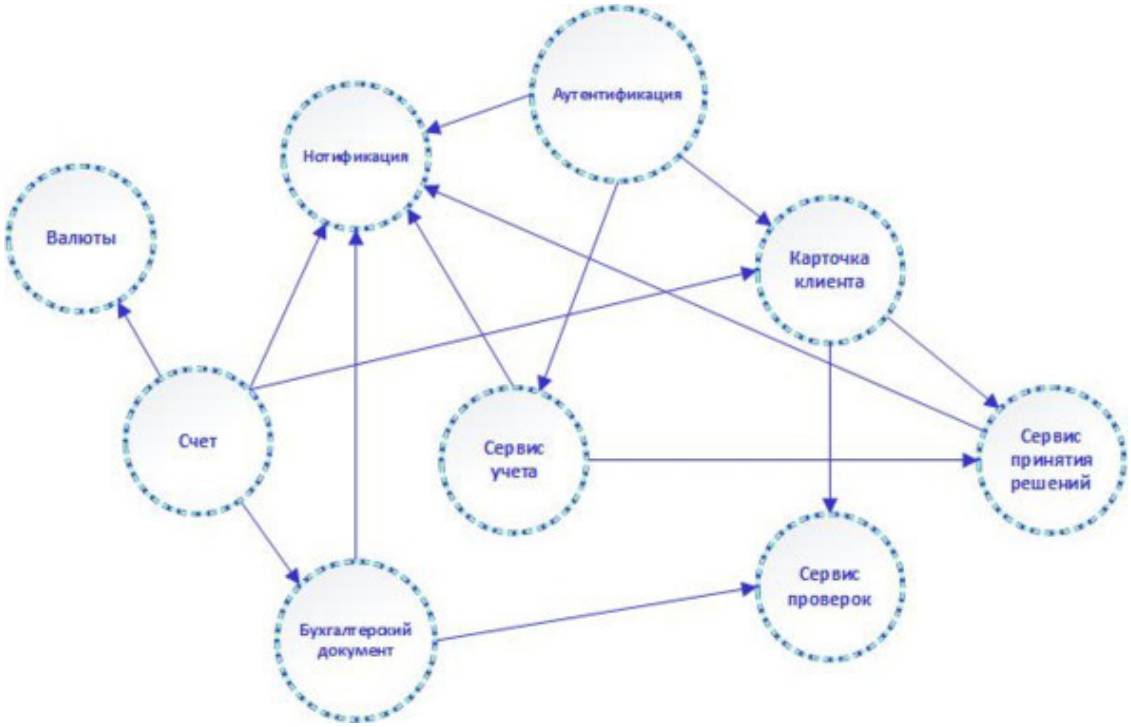

Рисунок 8. Пример концепции микросервисной архитектуры

Показанные на Рисунке 8 микросервисы и межсервисные взаимодействия представлены в качестве примера.

При проектировании решений на основе микросервисной архитектуры учитывались следующие характеристики и качества микросервисов:

• Трудоемкость. Разработку микросервиса должна вести одна команда, сформированная в соответствии с гибкими практиками разработки.

• Независимость. Локальные задачи решаются средствами микросервиса. Команда, реализующая микросервис, не должна ожидать исполнения подзадач (в составе данного микросервиса) другой командой. Возникновение ситуаций ожидания являлось признаком ошибок проектирования.

• Бизнес-ориентированность. Микросервисы должны решать конкретные прикладные задачи заказчиков.

• Простота интеграции. Для реализации взаимодействия микросервисов не требуется использования отдельных программных компонентов (по примеру рассмотренного выше ESB решения).

• Учет возможных ошибок. Принимая во внимание возможность возникновения ошибок при выполнении жизненного цикла информационной системы, при проектировании последней в парадигме микросервисной архитектуры следует учитывать потенциальную недоступность каждого микросервиса. При этом система должна реагировать таким образом, чтобы минимизировать вероятность общесистемного сбоя.

• Децентрализация данных. Отдельные микросервисы работают с независимыми источниками данных в рамках собственных моделей данных. Модели данных разных микросервисов развиваются независимо друг от друга.

На первый взгляд переход к практикам микросервисной архитектуры вносил существенные упрощения в разработку информационных систем, а также повышал наглядность их проектирования, принося следующие результаты:

• Возможность гибкого выбора технологий. При независимом исполнении и развитии микросервисов, их автономности от особенностей реализации контрагентов, каждый микросервис может разрабатываться с использованием собственных технологий, выбор которых диктуется потребностями заказчика, наличием в компании соответствующих компетенций и т. д. Необходимо поддерживать лишь исключительно API-совместимость с контрагентами.

• Независимость данных. Наличие собственных источников данных, а также использование собственных моделей данных исключает необходимость затрачивать существенные финансовые, временные и трудовые ресурсы на поддержание тяжеловесных шаблонов, таких, как каноническая модель данных.

• Отказоустойчивость. При проектировании системы с учетом минимизации влияния каждого микросервиса на стабильность работы системы в целом, показатели отказоустойчивости последней возрастают.

• Упрощение замены блоков функционала. В случае, когда бизнес-функции реализуются минимально зависимыми компонентами – микросервисами – их замена оказывает минимальное влияние на другие компоненты системы.

• Удобство и дешевизна масштабирования. При увеличении нагрузки на систему отсутствует необходимость в масштабировании всего программного комплекса (в отличие от эпох монолитных приложений и SOA), а возможно масштабировать только те микросервисы, на которые оказывается дополнительная нагрузка, что снижает технологические и финансовые риски.

• Соответствие создаваемой архитектуры организационной структуре предприятия. Возможным становится создание микросервисов, реализующих автоматизацию функционала, напрямую соответствующего должностным обязанностям подразделений предприятия. Таким образом уже на уровне концепции для микросервисной архитектуры реализуется закон Конвея.

• Снижение издержек при тестировании. При проектировании и разработке крупных промышленных информационных систем серьезным препятствием динамичному развитию являются потенциальные объемы регрессионного тестирования. В случае распределенной системы, построенной на основе минимально зависимых компонентов (микросервисов), объем потенциального регрессионного тестирования при развитии существенно снижается, что в свою очередь снижает финансовые и технологические риски для заказчика.

• Удобство развертывания. При обновлении отдельного микросервиса отсутствует необходимость обеспечивать развертывание всей информационной системы.

Указанный перечь характеристик и потенциальных преимуществ перехода к микросервисной архитектуре обеспечили повышенный интерес к данной архитектурной концепции в 2010-е годы. Ряд технологических гигантов уже в начале десятилетия начали перевод своих технологических платформ на микросервисную архитектуру. В качестве примера можно привести Uber.

Следует отметить, что компании, осуществлявшие переход к микросервисной архитектуре, столкнулись с рядом сложностей. Компания Uber посредством своих Интернет-ресурсов (https://eng.uber.com/microservice-architecture/, 23.07.2020) достаточно подробно рассматривала проблемы, с которыми ей пришлось столкнуться. По ходу решения выявленных проблем ИТ-ландшафт компании неоднократно претерпевал серьезные изменения, иногда включавшие в себя полный пересмотр достаточно крупных элементов данного ландшафта, а также деталей проектирования. На ряде популярных Интернет-ресурсов профессиональной направленности размещались статьи многих компаний, содержавшие выражения яркого эмоционального характера по адресу микросервисной архитектуры. Основной претензией, высказанной в отношении архитектурной концепции, были резко возросшая сложность ИТ-решений, запу-

танность интеграций, трудности в организации управления бизнес-процессами, сложности с сопровождением созданного решения, невозможность выработать эффективную релизную модель. Анализируя проблемы, с которыми столкнулись участники рынка при переходе к микросервисной архитектуре, можно отметить следующие основные примеры неудачной реализации новых архитектурных концепций, которые и привели к упомянутым проблемам:

• Построение «распределенного монолита». В профессиональном сообществе «распределенный монолит» стал устойчивым термином, характеризующим высокую связность компонентов решения, которая в условиях распределенной архитектуры и следующего за ней развертывания не только не решала проблемы взаимовлияния компонентов друг на друга, но и дополняла их сетевой латентностью при осуществлении удаленных вызовов.

• Слишком мелкая грануляция микросервисов. При крайне мелком дроблении создаваемой ИТ-системы на микросервисы возможна исключительно сложная топология ее развертывания. В подобном случае количество потенциальных зависимостей (даже на уровне API) между отдельными микросервисами может значительно усложнить развитие ИТ-системы, что в свою очередь негативно повлияет на показатели времени вывода продуктов заказчика на рынок, надежность и производительность создаваемого решения. Следуя терминам профессионального сообщества, на смену «зоопарку систем» приходил «серпентарий микросервисов».

• Слишком крупная грануляция микросервисов. Является обратной стороной предыдущего пункта и близка к ситуации «распределенного монолита». В качестве микросервисов выделяется несколько подсистем, которые содержат большое количество логики и, по сути, представляют собой полноценные ИТ-системы предыдущего архитектурного поколения, а не микросервисы.

• Большое количество точечных интеграций. При прямом взаимодействии микросервисов между собой возможна избыточная сложность построения интеграционных взаимодействий. В качестве примера можно привести обновление web-интерфейса приложения, требующего для выдачи данных с последующим представлением пользователю последовательного вызова 4—5 микросервисов. Обеспечить высокую производительность (необходимую для использования в удаленных каналах обслуживания) подобным образом спроектированного решения зачастую проблематично.

Представленные проблемы показывают высокую сложность перехода к микросервисной архитектуре и не позволяют считать ее саму по себе вариантом революционного перехода к новому поколению архитектур. Тем не менее, революция произошла. Ее обеспечили два следующих аспекта.

Первым стал продуктовый подход к архитектуре, позволивший эффективнее обеспечить ее применение в сочетании с гибкими методологиями разработки. Традиционно при проектировании архитектурных решений внимание уделялось информационным системам, но в рамках микросервисной архитектуры данное понятие потеряло свое предыдущее значение. Манифест Agile провозгласил ориентацию ИТ на решение проблем заказчиков, максимально быстрое создание минимально жизнеспособных продуктов (MVP), декларируя открытость ИТ миру. Архитектура не могла оставаться в стороне от подобных веяний, а потому также сменила фокус внимания на продукты, имеющие конкретное значение для заказчиков. Концепция продуктов в архитектуре будет рассмотрена в соответствующей главе. Отметим, что в случае микросервисной архитектуры было внесено предложение, получившее популярность благодаря успешной практике применения, проектировать новые ИТ-решения таким образом, чтобы микросервисы (или их группы) отвечали бы за автоматизацию продукта, представляющего ценность для заказчика. Таким образом, соответствующий микросервис (или группа микросервисов) могли бы разрабатываться действительно небольшой командой, сохраняя характе-

ристики трудоемкости, представленные выше. Микросервисы, отвечающие за продукт, могут поддерживать режим сильной связности между собой, при этом их взаимодействие с микросервисами, обеспечивающими автоматизацию логики других продуктов, ограничено и подчиняется принципам слабой связности. Продуктами для финансовой сферы, на примере которой происходит рассмотрение в настоящей главе, могут служить вклады, накопительные счета, кредиты, банковские гарантии. Пример продуктовой микросервисной архитектуры представлен на Рисунке 9.

На Рисунке 9 показаны два банковских продукта (кредит юридическому лицу и банковская гарантия), автоматизация которых осуществляется в соответствии с микросервисной архитектурой. Микросервисы, осуществляющие автоматизацию одного продукта, обладают множеством связей между собой, при этом количество связей между микросервисами, отвечающими за автоматизацию различных продуктов, минимизировано. Таким образом осуществляется движение к атомарности реализации соответствующей продуктовой функциональности.

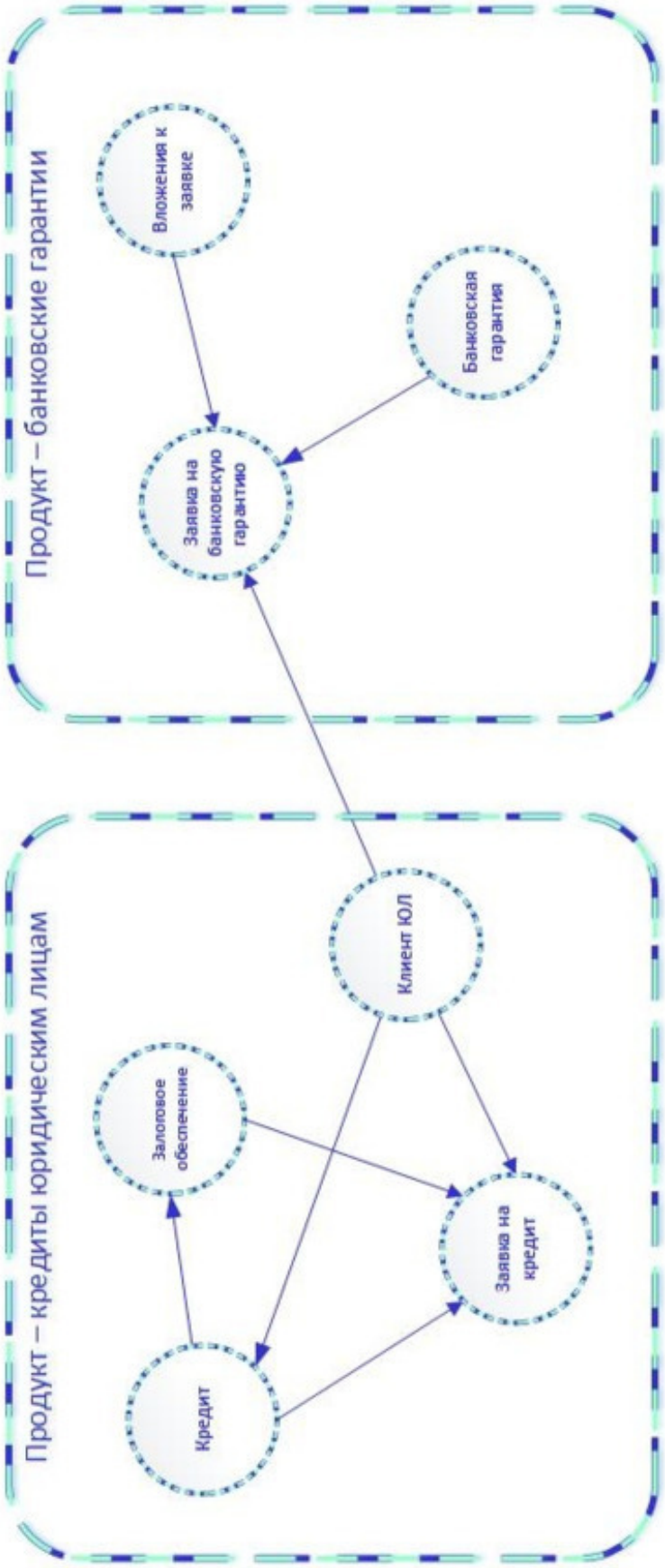

Рисунок 9. Пример продуктовой микросервисной архитектуры

Надо отметить, что внедрение продуктового подхода в концепции микросервисной архитектуры упростило организацию атомарных единиц развертывания (под которыми понимаются уже не отдельные микросервисы, а микросервисные группы, отвечающие за автоматизацию продуктовой логики), но концепция интеграции осталась потенциально сложной. Также необходимо отметить сложность современных продуктов цифрового мира. Отдельные группы микросервисов могут (в случае автоматизации сложного продукта) представлять собой программные комплексы, соответствующие информационным системам предыдущего поколения. И в таком случае упомянутые выше проблемы, например, «распределенный монолит», могут остаться для них актуальными. Поэтому вторым аспектом, обеспечивающим революционность перехода к микросервисной архитектуре, стало внедрение в нее концепций событийно-ориентированной архитектуры.

Нельзя не упомянуть, что само понятие событийно-ориентированной архитектуры (event-driven architecture, EDA) и ее концепции старше, нежели микросервисная архитектура. В свое время методики EDA конкурировали с SOA за право считаться наиболее распространенными и применимыми при автоматизации различных областей человеческой деятельности, но не приобрели аналогичной популярности. Более того, зачастую концепция интеграции компонентов информационных систем и систем между собой посредством событий (основополагающая методика EDA) подменялась на практике обычным асинхронным обменом, представлявшим собой отложенные команды на исполнение. Можно сказать, что массовый переход к микросервисной архитектуре стал «вторым дыханием» для EDA. Решение таких задач как слабая связность компонентов между собой, снижение числа прямых интеграций, масштабируемость и надежность программных комплексов, независимость исполнения бизнес-функций, не терявших своей актуальности при переходе к микросервисной архитектуре обеспечивалось практиками EDA. Рассмотрим более подробно применение EDA в микросервисной архитектуре.

Основой взаимодействия микросервисов между собой является событие. Под событием понимается изменение состояния микросервиса, оказывающее влияние на состояние информационной системы или ИТ-ландшафта в целом. Соответственно, реализуемые микросервисы должны содержать прослушивающие процессы, позволяющие обеспечить реакцию на публикуемые события. Указанный подход позволяет минимизировать непосредственные взаимодействия микросервисов между собой (публикуемое событие может означать, что его прочитают несколько подписчиков, которые настраивают прослушивающие процессы в соответствии с бизнес-задачами, при этом публикатор события может не обладать информацией о том, кто конкретно прослушивает публикуемое им событие). При этом публикации событий и реакция на них производятся асинхронно, что снижает нагрузку на систему в целом и отдельные микросервисы. Все отмеченное не означает запрета на синхронные и прямые взаимодействия, но они должны быть минимизированы (например, ограничены областью автоматизации конкретного продукта, представляющего ценность для заказчика). При этом события обладают следующими свойствами:

• События должны быть «тонкими». Исключается передача в составе одного события больших объемов данных, создающих избыточную нагрузку на инфраструктуру.

• Фиксация событий. События отражают уже случившиеся изменения в микросервисах или системах, а не являются отложенными командами на исполнение.

• Независимость сериализации. Поскольку микросервисы (группы микросервисов) представляют собой независимые единицы развертывания, то взаимодействие между ними предполагает сериализацию данных. Возможность выбора различных технологий реализации микросервисов диктует необходимость отделения правил и технологий сериализации событий от микросервисов – в противном случае гибкость ИТ-ландшафта будет нарушена.

• Наличие реестра схем. Общая структура событий и возможность быстрого получения ее, в том числе во время исполнения разработанных решений, позволит снизить число ошибок при выполнении микросервисов и информационных систем в течение их жизненного цикла.

• Валидация событий. При взаимодействии микросервисов и информационных систем между собой посредством EDA возможна публикация событий с некорректной структурой, что особенно актуально для распределенной мелко гранулированной информационной системы. Структуры событий должны проверяться на соответствие схемам для обеспечения надежности.

• Бизнес-ориентированность. События должны отражать значимые изменения в состоянии микросервисов и информационных систем. В противном случае в архитектуре возникнут риски создания большого количества незначимых событий, не несущих ценности, но снижающих прозрачность ИТ-ландшафта и создающих избыточную нагрузку на инфраструктуру.

• Мониторинг и отладка. Событийная инфраструктура должна обеспечивать мониторинг создаваемых событий и реакции на них, отладку выявляемых ошибок на максимально раннем этапе.

• Возможность самопотребления. Микросервисы могут сами реагировать на опубликованные ими события. Такая возможность должна обеспечиваться при реализации.

Пример решения, выстроенного в соответствии с концепциями микросервисной архитектуры, продуктового подхода и EDA, представлен на Рисунке 10.

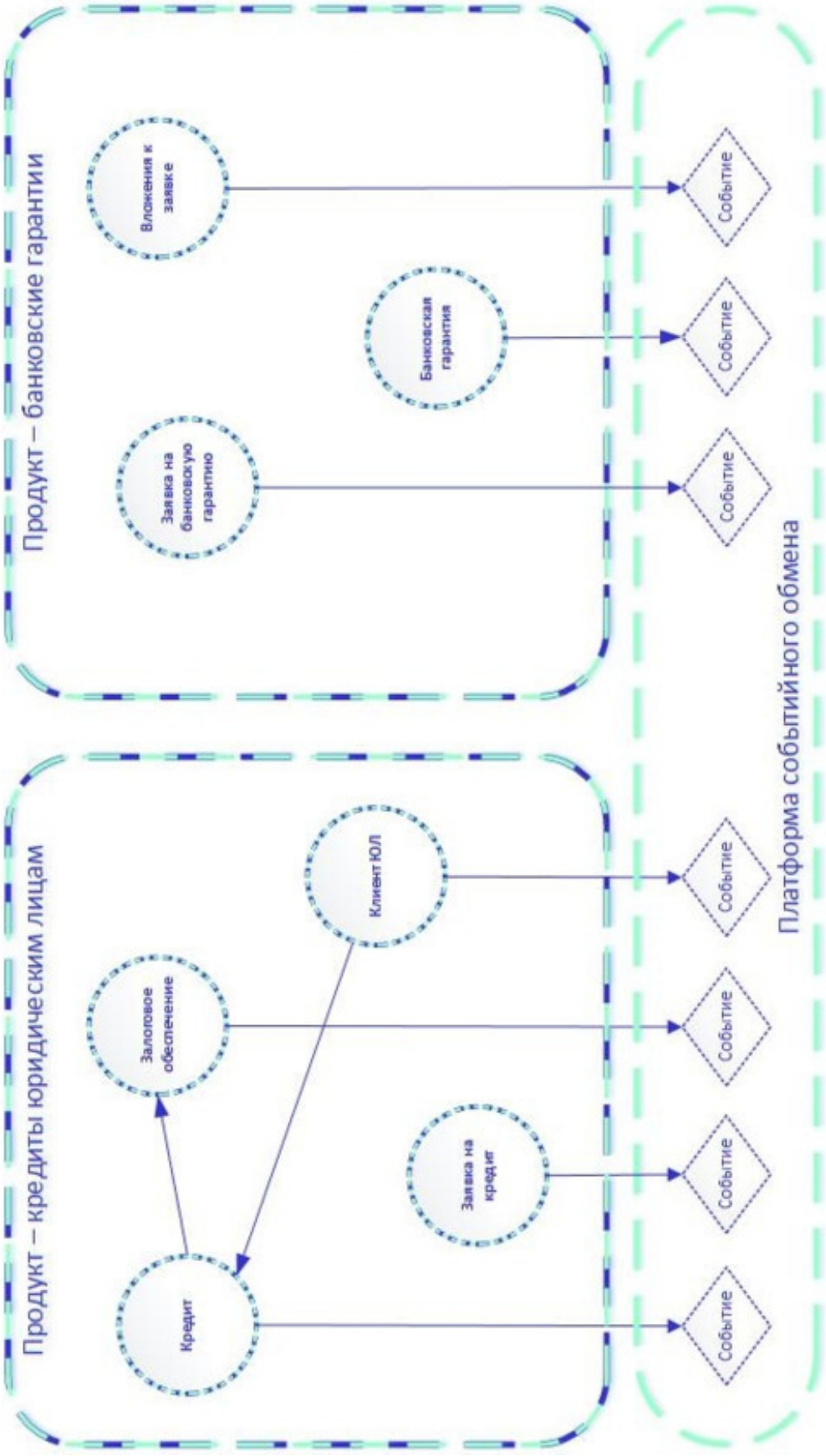

Рисунок 10. Пример продуктовой микросервисной архитектуры в сочетании с EDA

На Рисунке 10 показаны два банковских продукта, автоматизируемых в соответствии с принципами микросервисной архитектуры и EDA. Основной интеграционный поток осуществляется посредством публикации событий, управление которыми осуществляет платформа событийного обмена, представляющая собой внешний по отношению к микросервисам технический компонент. Отличие указанной платформы от ESB-решений эпохи SOA заключается в строго техническом характере решения и «пассивном» положении платформы в ИТ-ландшафте – она не оказывает непосредственного концептуального влияния на интегрируемые микросервисы. При этом прямые интеграционные взаимодействия между микросервисами также присутствуют, но минимизированы.

Сочетание микросервисной архитектуры, продуктового подхода и EDA позволило многих компаниям провести кардинальную трансформацию своего ИТ-ландшафта, осуществив новый революционный переход, и значительно повысить качество предлагаемых услуг, обеспечить высокую скорость внесения изменений и вывода продуктов на рынок. При этом на новый качественный уровень вышли производительность и надежность их решений. Соответствующие архитектурные подходы используются в LinkedIn, Uber, Netflix, Monzo Bank и других технологических лидерах современности.

Нельзя не отметить важного следствия случившихся революционных переходов – все они шли в рамках тенденции по упрощению разработки программных компонентов и их максимального приближения к непосредственной ориентации на решение бизнес-потребностей. Вместе с этим существенно выросла сложность проектирования архитектуры новых решений: дробление на микросервисы, выделение продуктовых границ на уровне ИТ-ландшафта, определение изменений состояния микросервисов, значимых с точки зрения публикации событий и т. д. Все это требует совершенно нового уровня понимания архитектуры создаваемых решений, а также глубокого погружения в предметную область. Также существенно выросли требования к техническому обеспечению выполнения создаваемых решений. Например, платформа событийного обмена, выполняя технические функции, должна обеспечивать исключительно высокие производительность и надежность, необходимые для современных решений, применимых в цифровом мире. При этом разработка каждого конкретного программного компонента существенно упростилась.

Но следует обратить внимание и на другой важный аспект революционного перехода – на mindset. По ходу настоящей главы было проиллюстрировано развитие подходов к архитектуре, показано, какой скачок произошел за двадцать лет. На фоне многих других отраслей человеческой деятельности, также осуществлявших свои революционные и эволюционные переходы, ИТ развивается очень быстро. И mindset людей, задействованных в ИТ, далеко не всегда успевает за изменениями. Вопросы развития архитектуры являются наглядным показателем данного явления.

Многие компании, активно развивающиеся в различных направлениях человеческой деятельности, имеют давнюю по меркам ИТ историю, что непосредственно сказывается на уровне их автоматизации. В России крупнейшие корпорации осуществляют автоматизацию своей деятельности 25—30 лет, в США и Западной Европе этот процесс даже более длителен. По ходу развития возникает ситуация сосуществования ИТ-решений различных поколений: рядом сосуществуют монолитные решения, многие из которых разработаны с использованием устаревших технологических стеков, сервис-ориентированные решения, есть наработки с использованием микросервисной архитектуры. Нередко в профессиональном экспертном сообществе поднимается вопрос, зачастую инициируемый и заказчиками решений по автоматизации, который заключается в том, какой путь выбрать в части архитектурных преобразований: эволюционный или революционный. В случае перехода организации, обладающей большим объемом унаследованного (legacy) ИТ-ландшафта вопрос можно конкретизировать

следующим образом: стоит ли дробить системы предыдущего поколения на микросервисы, либо стоит создавать новый ИТ-ландшафт фактически с нуля.

Первый из представленных способов видится более простым, управляемым и (ключевое заблуждение) дешевым. К сожалению, большая часть комментариев негативного характера в адрес микросервисной архитектуры, о которых говорилось выше, обусловлена именно приоритетом данному варианту развития (эволюционному). На первый взгляд, ситуация кажется простой: провести анализ созданных ранее решений, выделить в них те элементы, которые могут представлять собой микросервисы, а далее провести реализацию очерченного функционала в соответствии с намеченным планом и на новом технологическом уровне. При этом зачастую к новым решениям применяется mindset прошлого – типовым вариантом разбиения информационных систем на микросервисы стала реализация функциональных подсистем в формате отдельных микросервисов. При этом подсистемы зачастую были сильно связаны между собой, содержали большое количество перекрестных вызовов, так называемые «божественные» объекты и функции (содержащие избыточное количество данных и функционала соответственно). Результатом обычно был рассмотренный по ходу настоящего раздела «распределенный монолит». Снижалась производительность систем, их надежность и управляемость ИТ-ландшафта организаций в целом. Релизные модели и циклы предыдущих поколений не позволяли эффективно управлять жизненным циклом изменений, объемы регрессионного тестирования существенно возрастали. Для устранения возникавших проблем приходилось расширять штат специалистов, решать возникшие проблемы механическим увеличением трудозатрат; альтернативой становился революционный переход (при этом существенные временные, трудовые и финансовые ресурсы уже были вложены в попытку перехода эволюционного).

Революционный переход предполагал создание нового ИТ-ландшафта параллельно существующему. С целью исключения нарушения стабильности функционирования производственных и бизнес-процессов текущие информационные системы продолжали выполнять задачи, в то время как по мере создания и развития нового ИТ-ландшафта в парадигме микросервисной архитектуры новый функционал создавался (переводился) в рамках него. Перевод осуществлялся по бизнес-функциям, а не по структурам предыдущих архитектурных поколений. По итогам замещения функционала, выполняемого унаследованным ИТ-ландшафтом, последний выводился из эксплуатации. Потенциальные проблемы mindset наблюдались и в случаях революционного перехода: соблазн следовать шаблонам прошлого, идти проторенным путем часто превалировал над новыми концепциями. Но при создании нового ИТ-ландшафта «с нуля» выявление проникновения практик предыдущих поколений производилось на более ранней стадии, нежели в случае эволюционного перехода, в связи с чем могло оперативно устраняться. Именно таким путем шли лидеры технологического рынка.

Таким образом, варианты революционного и эволюционного развития архитектуры можно уподобить Сцилле и Харибде из «Одиссеи»: проход рядом со скалой Сциллы грозил Одиссею и его спутникам гибелью части экипажа, Харибда же – общей гибелью всех мореходов. Именно поэтому главным аспектом архитектурного перехода, развития архитектуры являлся и является mindset, который должен быть направлен строго на революционное развитие.

Архитектура и открытый код

Ранее уже отмечалось, что использование решений с открытым исходным кодом является одной из ключевых тенденций развития современной архитектуры. Следование данной тенденции позволяет существенно ускорить разработку новых ИТ решений, а также повысить их качество, но и предъявляет новые требования к работе архитектора.

17 сентября 1991 года Линус Торвальдс опубликовал в сети исходный код ядра операционной системы Linux, что считается началом эпохи программных продуктов с открытым исходным кодом. Долгое время программное обеспечение с открытым исходным кодом, имевшее свободно распространяемые версии, считалось уделом узких групп энтузиастов, его применение в реальной промышленной среде представлялось нецелесообразным ввиду опасений в части надежности, безопасности, сопровождения и т. д. Тем не менее, данное направление программного обеспечения оказалось востребовано стартапами (как утверждали недоброжелатели, исключительно из-за фактора отсутствия лицензионных платежей). Но и по мере успеха ряда стартапов и превращения их в технологические гиганты (например, Meta Platforms, ранее Facebook), интерес уже лидеров технологического рынка к решениям на базе открытого кода не только не снижался, но и сами компании стали активными участниками процесса создания новых решений с открытым исходным кодом – выше уже приводился пример распределенной СУБД Apache Cassandra (исходно проект создан в Facebook). Можно утверждать, что существует связь между гибкими практиками разработки, решениями с открытым исходным кодом и технологическим прорывом.

В XVII веке итальянский философ и экономист Антонио Серра в работе «Краткий трактат о средствах снабдить в изобилии золотом и серебром королевства, которые их не добывают» отметил: «Если ты хочешь узнать, какой из двух городов богаче, определи, каким количеством профессий владеют его жители. Чем больше профессий, тем богаче город». Приведенный тезис можно считать первым направлением в оценке влияния разделения труда на общую эффективность производства. Развитие данных наработок было произведено Адамом Смитом в труде «Исследование о природе и причинах богатства народов». Результатом стала теория разделения труда как основы повышения эффективности производственных цепочек. В дальнейшем данный тезис практически не оспаривался как сторонниками учения Смита, так и противниками. Необходимо отметить, что для ИТ данная теория также справедлива. Развитие и использование решений с открытым исходным кодом является ее наглядным подтверждением.

Безусловно, отсутствие лицензионных платежей было важным фактором, обусловившим использование открытых решений в стартапах. Но сводить все к фактору «бесплатности» в корне неверно. «Бесплатных» технологий не существует: свободно распространяемое программное обеспечение исходно требовало значительно больших усилий для достижения требуемых показателей работоспособности, нежели «закрытое» (vendor-lock), в случае которого можно было полагаться на экспертизу сотрудников компании поставщика. Но именно этот аспект, исходно казавшийся негативным, оказался серьезным преимуществом. За поставляемым солидными компаниями программным обеспечением стояли истории успеха, лучшие практики, «золотые» топологии, крупные и длительные проекты внедрений. В случае необходимости получения быстрых результатов и минимально жизнеспособного продукта, поставка которого заказчикам и/или инвесторам становилась вопросом выживания стартапов, во всех перечисленных выше атрибутах «закрытого» программного обеспечения не было практического смысла. Гораздо более важными аспектами, характеризующими технологии, были возможности максимально быстрой адаптации под нужды компании – добавление нового функционала, исправление ошибок, отработка новых конфигураций и т. д. В этих аспектах «открытое» программное обеспечение априори превосходило «закрытое». С одной стороны,

у компании-стартапа была возможность доработки технологий с открытым исходным кодом под максимально полную технологическую синергию с решаемыми задачами. С другой стороны, заказчик получал решение именно своих конкретных задач, а не реализацию и повторение шаблонов прошлого. Указанные аспекты приносили открытому программному обеспечению соответствующую популярность. С ростом популярности происходило расширение и углубление разделения труда в разработке технологий.

Предположим, что существовало программное обеспечение X, позволяющее решать технологические задачи, востребованные стартапами. Каждый стартап, используя X для автоматизации своих нужд, добавлял в него что-то свое в части функционала. Формировавшиеся правила развития решений с открытым исходным кодом рекомендовали (а в случае ряда лицензий и требовали) публиковать вносимые изменения, дополнительный функционал в открытом доступе, развивая X. X, обретая дополнительный функционал, получал потенциал дополнительной популярности среди стартапов. По мере превращения стартапов в технологические гиганты рос и соответствующий потенциал популярности X, увеличивалось число архитекторов и разработчиков, использовавших X в своих решениях. Иллюстративно это показано на Рисунке 11.

Стартапы, ряд которых перерастает в технологические гиганты, используют для автоматизации своих потребностей программное обеспечение X с открытым исходным кодом. При этом география компаний распределенная, они работают по всему миру и на регулярной основе публикуют изменения, производимые в X. Необходимо отметить, что при развитии стартапов до уровня технологических лидеров они занимали на рынке ниши, которые оставались незанятыми (или занятыми в незначительном объеме) компаниями традиционных направлений человеческой деятельности. При этом и требования к ним предъявлялись новые. Например, упоминавшийся пример технологического гиганта Meta Platforms (ранее Facebook), предложившего миру глобальную социальную платформу, ставшую не только средством общения, формирования и продвижения социальных связей, но и площадкой ведения бизнеса, предполагал работу с принципиально новым кругом пользователей и партнеров, а также предложение и поддержку новых форматов работы. Данный пример предполагал качественно новые требования к используемому программному обеспечению, в результате специалисты компании активно развивали технологические платформы, публикуя вносимые изменения. В итоге программное обеспечение с открытым исходным кодом приобретало функционал, качественно отличавший его от «закрытых» решений.

Рисунок 11. Стартапы и открытый код (X)

На Рисунке 11 показано, как стартапы, работающие в самых разных областях человеческой деятельности, создающие новые ее направления, используют и развивают программное обеспечение с открытым исходным кодом X, составляя глобальную технологическую цепочку в рамках концепции разделения труда.

Более того, по мере роста технологических гигантов компании традиционных отраслей экономики также стремятся в открывающиеся и/или создаваемые новые рынки. И в этом случае программное обеспечение X уже покрывает часть требуемого функционала, либо же обеспечивает выполнение нефункциональных требований. Например, созданная компанией Meta Platforms (ранее Facebook) распределенная СУБД Cassandra (ныне Apache Cassandra) исходно поддерживала децентрализованное исполнение и нереляционную логику, что на момент выхода ряда компаний на рынки, требовавшие распределенности, либо не поддерживалось популярными «закрытыми» СУБД, либо было побочным функционалом последних. Компании традиционных отраслей экономики (например, Goldman Sachs Group) также включаются в процесс работы над открытым кодом и зачастую публикуют производимые в нем изменения, тем самым дополнительно повышая функциональность и качество X. Итогом подобного стремительного развития становится цепочка разделения и углубления труда, недостижимая «закрытыми» решениями, какими бы крупными корпорациями последние ни развивались. В соответствии с базовыми экономическими законами это обеспечивает и большую эффективность развития решений с открытым исходным кодом. Неудивительно, что данный класс программного обеспечения исключительно быстро прошел путь от удела небольших групп энтузиастов до флагманского направления ИТ.

На сегодня программное обеспечение с открытым исходным кодом используется крупнейшими компаниями, работающими во всех отраслях человеческой деятельности. И это является архитектурно значимым фактором, в корне меняющим характер деятельности архитекторов, а также само развитие архитектуры.

В эпоху господства «закрытого» программного обеспечения архитектор, работавший в компании-заказчике программных решений, при проектировании информационных систем учитывал ключевые решения компаний-поставщиков, имевшиеся с ними контракты, ключе-

вые топологии, но при этом зачастую не погружался глубоко в особенности функционирования поставляемых комплексов. Многие знания были и закрыты от него, составляя богатство служб поддержки премиального уровня. Требования к архитектору, работавшему в компании, реализовывавшей решения на базе «закрытого» программного обеспечения, например, в компании-партнере поставщика соответствующего ПО, были во многом аналогичными. Результатами проектирования были программные комплексы, учитывавшие известные особенности и лучшие практики внедрения поставляемого программного обеспечения, дополненного кодом, разработанным на его основе.

Проектирование и разработка решений на основе программного обеспечения с открытым исходным кодом значительно отличаются. Данное программное обеспечение предоставляет широкий набор функциональных возможностей, обеспечивает выполнение широкого спектра нефункциональных требований, но нуждается в «тонкой» настройке, кроме того, его конфигурация выбирается в соответствии с такими особенностями создаваемого решения, как характер исполнения, число, характеристики и классы пользователей, категории функционала, подлежащего автоматизации, принципы хранения и обработки данных, а также многими другими. Архитектор при проектировании новой информационной системы должен учитывать все вышеупомянутые особенности, а также возможности использования открытого программного обеспечения, выбирать топологию последнего соответствующим образом. Безусловно, архитектор не может быть специалистом в деталях используемого программного обеспечения, но ключевые для создаваемого решения детали он обязан как знать, так и учитывать в архитектурном решении, делать на них акцент, задавая тем самым путь команде разработки. Если ранее недоброжелатели сравнивали работу архитектора с отрисовкой геометрических фигур («квадраты», «кубы»), серьезно упрощая ее, но используя в качестве основы для недобросовестной критики собственно результаты работы архитектора, то современные архитектурные решения содержат гораздо более глубокий уровень проработки, выбивающий почву из-под ног подобных критиков.

Необходимо рассмотреть и другое важное обстоятельство. Ранее уже отмечалось, что в процессе революционных переходов от традиционных монолитных решений к современной микросервисной архитектуре с учетом продуктового подхода и принципов EDA проявилась тенденция на упрощение разработки отдельных программных компонентов с одновременным увеличением сложности проектирования информационных систем. Наряду с разрабатываемыми программными компонентами (микросервисами) используется программное обеспечение с открытым исходным кодом: СУБД, движки управления бизнес-процессами, кэширующие элементы, потоковые платформы и т. д. Ранее (в эпоху монолитных приложений и SOA) фреймворк разработки программного кода имел важное, если не определяющее значение. На сегодня микросервисы могут разрабатываться на любом языке программирования, с использованием любого подходящего фреймворка. И гораздо более важное значение приобретает не выбор языка программирования, но совместимость и полная поддержка всего функционала используемого открытого программного обеспечения. Последнее, как и практически любое направление человеческой деятельности, развивается неравномерно. Полноценная совместимость и поддержка функционала в экосистеме Java может опережать аналогичную для. NET и наоборот. И здесь также крайне важна роль (и полномочия в компании) архитектора, закладывающего соответствующий технологический выбор.

Соответственно, и развитие архитектуры направлено не только в сторону расширения функционала и повышения удобства фреймворков программирования, не только в сторону добавления функционала, конфигураций, отказоустойчивости открытых программных комплексов, но и на достижение их синергии при создании новых решений.

Использование открытого кода, увеличение роли стартапов, превращение последних в технологических гигантов привело также к фундаментальным изменениям в mindset всего

рынка ИТ. Важную роль здесь играет и программное обеспечение с открытым исходным кодом, которое стало одним из ключевых факторов (наряду с культурой стартапов) в формировании так называемых открытых организаций. Последние описаны в книге Джима Уайтхерста (James M. «Jim» Whitehurst) «Открытая организация: Страсть, приносящая плоды» (2015, ISBN 978-5-9693-0405-5).

Целью настоящей книги не является изложение и разбор аспектов функционирования открытой организации. Важнее отметить другое: стартапы, превращаясь в технологические компании, сохранили культуру работы с открытым кодом, которая во многом формируется в рамках взаимного плодотворного влияния друг на друга открытого кода и общей производственной культуры компании-стартапа, предполагающей определенную свободу сотрудников. Успехи технологических гигантов, в короткое время выросших из стартапов, вдохновили на соответствующую трансформацию компании, имеющие давнюю историю работы. При этом трансформация осуществлялась не только в технологических компаниях, но и в тех организациях, непосредственная деятельность которых осуществлялась в других направлениях человеческой деятельности. По ходу данной книги отмечалось, что для современности характерно взаимовлияние ИТ и мира, превращающее все компании, которые развиваются с целью сохранения собственной конкурентоспособности, в технологические компании, оказывающие услуги в различных сферах бизнеса. Наглядным примером такой трансформации является «Сбербанк России», который предоставляет на основе собственной технологической платформы, созданной по гибким методологиям, целую экосистему услуг: финансовые, транспортные, деловые, медиа и многие другие. Таким образом, трансформация становится актуальной для всех мировых компаний. Происходит повсеместный переход к работе по гибким методологиям, цифровизация отраслей деятельности, открытие и занятие новых рынков, меняется mindset. И особенно важное влияние оказывает открытый код. Разберем это влияние в контексте архитектуры, основываясь на ключевым тезисах книги Джима Уайтхерста, посвященных открытой организации:

• Мотивация. Достаточно давно зафиксирован следующий психологический аспект: совместная работа служит делу сплочения коллектива, в результате чего все участники рабочего процесса начинают работать более эффективно, повышая эффективность всей производственной цепочки в целом. Наиболее оптимальным проявлением совместной работы является значимость вклада каждого члена команды. Сложно придумать более наглядный пример совместной работы, нежели развитие решений с открытым кодом: команды профессионалов со всего мира вовлечены в данный процесс, каждый может публиковать вносимые изменения, включаясь в состав тех, кто развивает ведущие мировые технологические решения. Практика учета всех публикаций позволяет отметить вклад каждого, про эффективность же глобальной цепочки разделения труда уже говорилось выше. Отметим, что одним из основополагающих пунктов манифеста Agile являлся тезис о том, что лучшие технологические и архитектурные практики рождаются у самоорганизующихся команд. В масштабе глобальной цепочки разделения труда архитектор, во-первых, становится трендсеттером, который может и должен задавать и корректировать направление работы, во-вторых, он должен анализировать все те предложения и формирующиеся практики, которые идут от команд разработки, и учитывать их при создании архитектурных решений. Таким образом, роль архитектора существенно трансформируется. Если ранее он диктовал правила игры, но лишь высокоуровневые, то теперь он становится участником процесса, не только формируя базовые правила, но и учитывая наработки команд, а потому его решения должны быть гораздо более детализированными с соответствующей глубиной проработки. Следование новой роли требует расширения вовлеченности архитектора в процесс создания программного обеспечения, а также во все рабочие процессы.

• Меритократия. При анализе и выборе технологических решений используются принципы меритократии, в соответствии с которыми полномочия при принятии решений сораз-

мерны вкладу участников в развитие создаваемой или развиваемой информационной системы. В соответствии с данным тезисом, роль архитектора также меняется: принятие архитектурно-технологических решений исключительно на основе предшествующего опыта или наличествующих предубеждений категорически недопустимо. В обсуждение при принятии технологических решений включается широкий круг специалистов, принимающих участие в производственной цепочке (напомним, глобальной), учитываются практики, вырабатываемые командами и отдельными участниками развития. В стартапах подобная культура дала взрывной эффект, поэтому не было никаких оснований отказываться от нее при росте компаний до статуса технологических гигантов, а также при трансформации компаний традиционных отраслей человеческой деятельности. Практика же учета соразмерности вклада в общее дело весу вносимых предложений позволяет исключить возможность перехода обсуждения выбора технологического решения лишь в разбор большого числа «мусорных» предложений. Соответственно, в задачи архитектора входит обеспечение своего значимого вклада в дело создания и развития программного обеспечения, формирование соответствующего авторитета, в противном случае он превратится в технического писателя, задачи которого сведутся к фиксации предложений ведущих участников технологической цепочки. Отметим, что данный принцип работы побуждает сотрудников активнее участвовать в процессе создания ценности, ведь тогда их мнения по будущим направлениям развития будут иметь больший вес.

• Прозрачность принятых решений. Очевидным следствием вышеперечисленных пунктов является прозрачность архитектурных решений для всех участников процесса создания программного обеспечения. В связи с этим следует подчеркнуть актуальность формирования архитектурных фреймворков, позволяющих вовлекать в обсуждение архитектуры максимально широкий круг участников. Создание и развитие в рамках общего решения артефактов, позволяющих вести дискуссии с узким кругом участников, становится малоэффективной потерей времени.

• Развитие новых направлений. При развитии новых направлений в открытой организации часто принято начинать с разбора сырых идей, не дожидаясь доведения последних до совершенства. Эффективнее становится обсуждать сырые идеи, проводя их шлифовку в рамках итераций обсуждения, доработки, прототипирования и т. д. Подобный подход, идущий в русле гибких практик, оказывает существенное влияние на развитие архитектуры. Архитектурные артефакты следует представлять команде и выносить на обсуждение как можно раньше. Коррективы, вносимые командой, могут оказаться исключительно значимыми, а потому получить их в качестве направляющей необходимо на ранних этапах развития идей. При выработке лучших идей и практик следует искать баланс между предложениями, основываясь на принципах меритократии. Максимально ранний и широкий учет мнений позволяет выявлять ошибки на ранних этапах проектирования, что повышает эффективность всей технологической цепочки.

Основываясь на вышеизложенном, следует отметить, что практики использования открытого кода (как и гибких практик) оказывают существенное влияние на создание архитектурных решений, а также на mindset архитектора. Также необходимо отметить, что при росте текущих цепочек разделения труда в части создания решений с открытым исходным кодом возможно дальнейшее повышение эффективности в части развития ИТ – на сегодняшний день данный потенциал расширения далеко не исчерпан. Исчерпание же соответствующего потенциала поставит вопрос о новом революционном переходе в сфере ИТ (при этом очевидно, что данный аспект необходимости революционного перехода не является единственным).

В завершение данного раздела автор выражает глубокую благодарность сообществу разработчиков и пользователей открытого кода, благодаря которым происходит столь интенсивное развитие ИТ. Особую благодарность мы выражаем фонду Apache Software Foundation, спо-

собствующему развитию огромного числа программных продуктов высшего мирового уровня, многие из которых автор и его коллеги использовали и продолжают использовать в своей непосредственной работе.

Архитектура и распределенность

Выше уже отмечалось, что распределенность является одной из ключевых тенденций развития архитектуры. При этом распределенность понимается в двух смысловых плоскостях: архитектура должна поддерживать распределенную структуру команд разработки, создающих новые ИТ-решения, а также распределенную топологию функционирования самих создаваемых ИТ-решений.

Как уже отмечалось, идеология развития программного обеспечения с открытым исходным кодом, создания решений на его основе позволили существенно увеличить цепочки разделения труда в сфере ИТ, что обеспечило кардинальное повышение эффективности реализации новых информационных систем. При этом при увеличении степени разделения труда растут риски отдельных участников цепочки, которые зависят от все большего числа поставщиков и потребителей. Данная тенденция характерна и для ИТ. Представим себе ситуацию, что информационную систему А разрабатывает команда из 100 человек. Но эти 100 человек не работают в одном офисе, выполняя заранее согласованные и спланированные блоки работ. Коллектив разбит на 10 команд по 10 специалистов, при этом каждая команда создает свой собственный блок функционала. Между блоками присутствуют зависимости как интеграционного (взаимодействие в рамках процессов обменов информацией), так и встраиваемого характера. Под последним традиционно понимается возможность включения результатов работы одной команды в рабочие блоки, создаваемые другой командой, в формате библиотеки. Современные технологии в части ИТ позволяют проводить тестирование функционала с помощью механизма «заглушек», эмулирующих функционал контрагентов, но указанный способ тестирования имеет известные пределы. Пример описываемой ситуации представлен на Рисунке 12.

Компоненты на Рисунке 12 показаны обезличено, чтобы не дополнять сложность распределенной структуры команды разработки сложностью предметной области. При этом Рисунок 12 является наглядной иллюстрацией растущих рисков каждой команды, участвующей в процессе разработки программного обеспечения, в рамках распределенной структуры, а также всего создаваемого решения: неготовность каждого блока функционала ставит под угрозу возможность изготовления всех блоков, с ним связанных, что в свою очередь отражается на реализации всего программного комплекса. Отметим, что принудительная синхронизация всех блоков работ между собой может привести к лавинообразному росту числа непроизводительных управленческих задач. Последнее прямо противоречит идеологии гибкой разработки, формулируемой в соответствии с манифестом Agile.

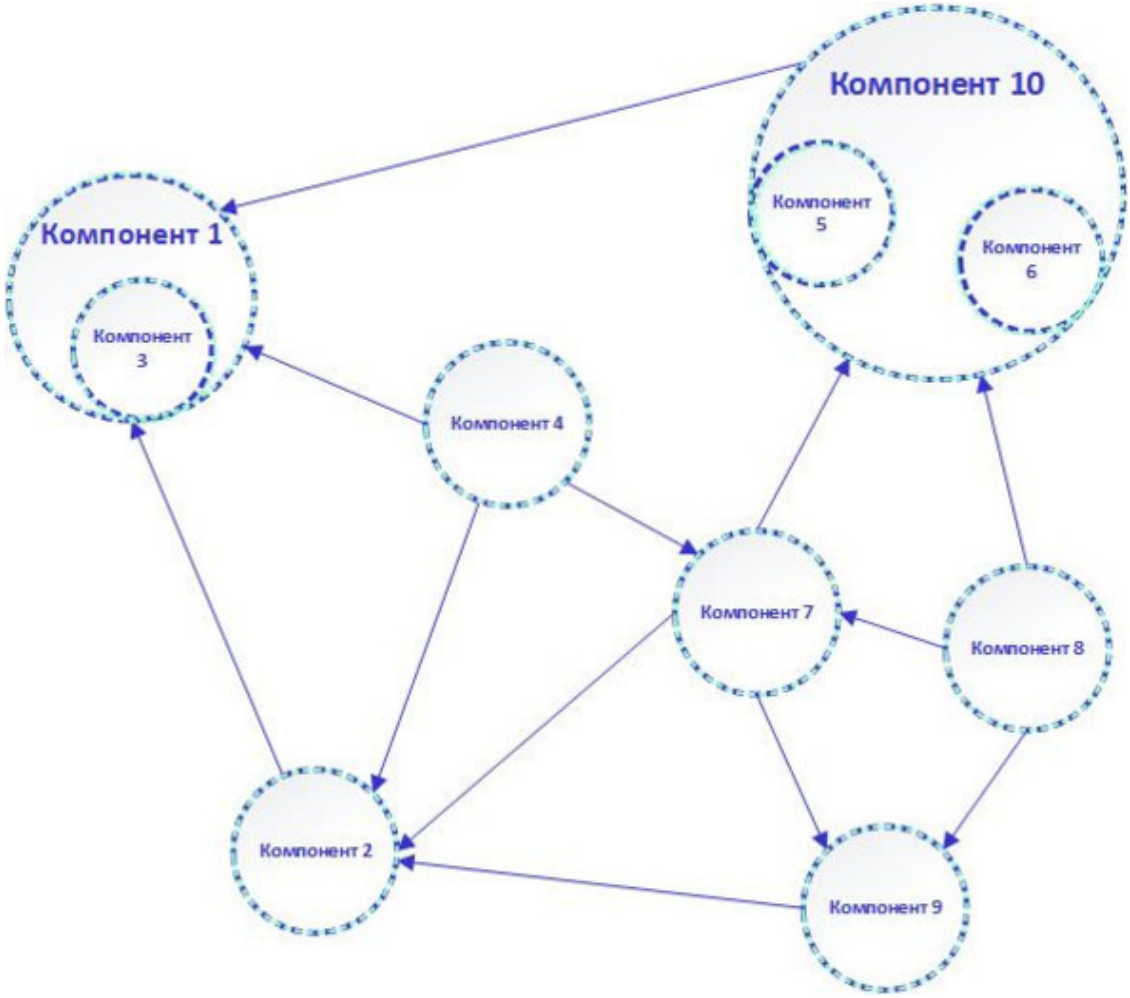

Рисунок 12. Распределенная команда разработки программного обеспечения

Решение представленной проблемы лежит не в плоскости управления, а в плоскости архитектуры. Конечно, возможно построить дорожную карту реализации всех компонентов, осуществлять контроль всех сроков исполнения, учитывая результаты итераций разработки, а также получаемую обратную связь от пользователей. Но указанный подход в последние годы получил негативную смысловую окраску, заслужив присвоение ему термина «микроменеджмент». Кроме того, подобный подход исключает возможность работы самоорганизующихся команд, противореча манифесту Agile. Каким же образом в данном случае может помочь архитектура, что требуется от архитектора?

По ходу настоящего изложения рассматривались революционные подходы к архитектуре, переход к микросервисной архитектуре с применением продуктового подхода и практик EDA, а также проблемы mindset. Использование лучших современных практик и должно быть применено в полной мере, mindset же должен служить соответствующим базисом. При функциональной декомпозиции создаваемого программного обеспечения работа распределенной команды разработки будет продолжать соответствовать Рисунку 12, исключая эффективность и независимость общей работы, а также увеличивая риски любого проекта. Иначе обстоит дело с применением продуктового подхода. Иллюстративно продуктовый подход, микросервисная архитектура и EDA в применении распределенной командой разработки показаны на Рисунке 13. Отметим, что серьезное упрощение, привносимое данными подходами, позволяет показывать на рисунке иллюстративные примеры и предметной области (как и ранее в книге, финансовой).

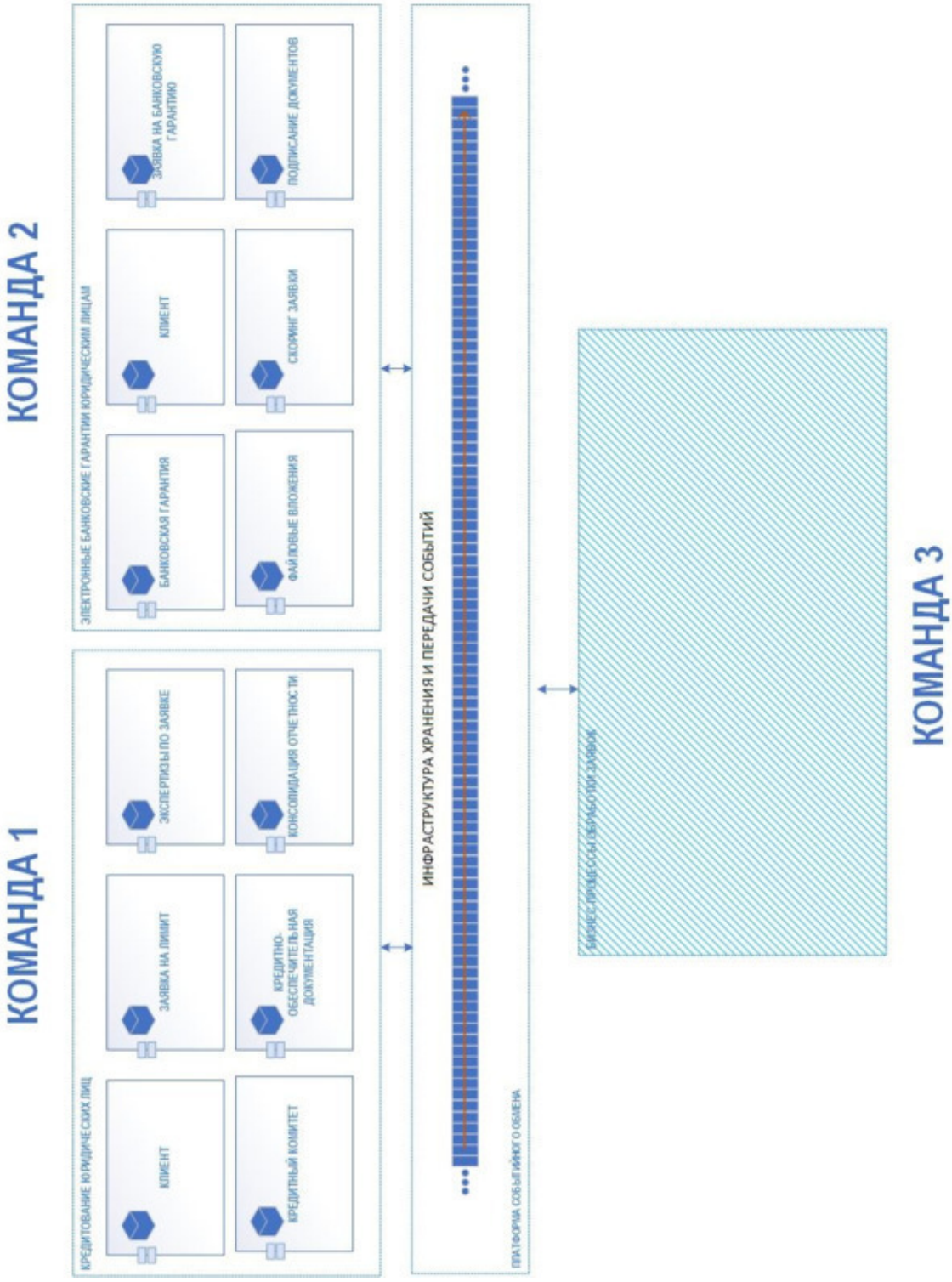

Рисунок 13. Применение микросервисной архитектуры, продуктового подхода и EDA распределенной командой разработки

На Рисунке 13 представлены 2 продукта, создаваемых различными командами разработки в составе общей распределенной команды. Общее решение создается в рамках обслуживания клиентов – юридических лиц, которым доступны два продукта: кредитование и элек-

тронные банковские гарантии. Одна команда разрабатывает продукт кредитование, вторая – электронные банковские гарантии. Каждое продуктовое решение создается в соответствии с парадигмой микросервисной архитектуры, при этом каждый микросервис является самостоятельной единицей развертывания. Взаимодействие микросервисов осуществляется посредством платформы событийного обмена и подчиняется принципам EDA. В чем же преимущество представленного подхода?

Каждый продукт представляет самодостаточную ценность для клиента, поэтому команды, создающие соответствующие продуктовые решения, могут независимо взаимодействовать с клиентами, представителями заказчика и иными заинтересованными лицами, демонстрируя результаты своей работы и получая необходимую обратную связь. При этом команды выбирают решения по структуре своих продуктов таким образом, чтобы исключить зависимость от контрагентов. Платформа событийного обмена, как отмечалось ранее, носит технический характер (в отличие от ESB решений), а потому необходимые настройки уровня хранения и передачи событий могут осуществляться членами команд, например, DevOps-инженерами. Ранее отмечалось, что различные продуктовые команды могут работать в соответствии с собственными моделями данных, не осуществляя синхронизацию последних между собой. На Рисунке 13 данная возможность проиллюстрирована наличием микросервисов «Клиент» в каждой продуктовой области. Поскольку речь идет о разных продуктах, то и команды, их создающие, развивают структуры данных и методы их обработки независимо друг от друга.

Отметим также, что бизнес-процессы обработки заявок на различные продукты могут содержать много общих (с точки зрения структуры бизнес-процесса) элементов. Кредитных продуктов и электронных банковских гарантий это касается в полной мере. Поэтому целесообразно выделить блок управления бизнес-процессами, конкретные элементы которых могут исполняться микросервисами различных продуктовых областей, при общей структуре самих шаблонов процессов. Взаимодействие посредством событийного обмена вносит свою долю независимости в данное решение. Добавим, что на Рисунке 13 само решение по управлению бизнес-процессами представлено обезличено, поскольку конкретика данного вопроса будет рассмотрена в соответствующем разделе.

Таким образом, различные продуктовые группы общего решения могут создаваться независимыми командами, в том числе с территориально распределенной структурой. При этом блоки управления бизнес-процессами также независимы от продуктовых областей. По мере усложнения решения будет расти число независимых продуктовых областей, вовлекая все больше команд в развитие решения, увеличивая тем самым цепочку разделения труда, но сохраняя независимость отдельных команд и минимизируя их риски с точки зрения взаимозависимости.

Особого внимания заслуживает тот факт, что для приведенной организации работ первичными являются не управленческие решения (которые также имеют место, пусть и не в столь явном качестве, как в традиционной парадигме), а архитектурные. Соответствующим образом формируются и требования к работе архитектора. Корректным образом спроектированная архитектура решения, задание ключевых направляющих дальнейшего развития и расширения, предоставление адекватного поля самоорганизации командам позволяют существенно повысить эффективность разработки новых информационных систем, а также составляющих их продуктов. Вместе с тем, ошибка, допущенная на этапе проектирования и не выявленная на ранних стадиях работ, может вернуть создаваемое ИТ-решение к ситуации Рисунка 12: распределенные команды начинают зависеть друг от друга, критически растут риски процесса создания решения, цепочка разделения труда оказывается неустойчивой. Данный пример является наглядной иллюстрацией ранее заявленного тезиса: при упрощении разработки каждого отдельного программного компонента сложность их архитектурного проектирования

возрастает, при этом цена архитектурной ошибки также растет. Нельзя не упомянуть в данном контексте известный принцип Альберта Эйнштейна: «Упрощать сложно, усложнять легко».

Нельзя не отметить тот факт, что вопрос корректного проектирования предъявляет требования к тому, что считать продуктом с точки зрения архитектуры – данный термин может приобретать иное значение, отличающееся от вводимого в гибких методологиях разработки. Подобное отличие нельзя считать некорректным – каждый смысловой уровень может давать собственные расшифровки терминов, актуальные для соответствующего применения. Далее вопросу продуктов с точки зрения архитектуры будет посвящен специальный раздел.

Одновременно с вопросом определения продукта представленный выше разбор поднимает вопрос обязательности с точки зрения современной архитектуры формирования кросс-функциональных команд, имеющих в своем составе специалистов, обладающих компетенциями для независимой работы. В случае отсутствия в команде DevOps-инженера, обладающего навыками работы с платформой событийного обмена, такая команда оказалась бы в зависимости от привлечения сторонних компетенций, что возвращало бы ситуацию к временам SOA.

При соблюдении озвученных выше принципов команды, представленные на Рисунке 13, могут работать независимо друг от друга и располагаться территориально таким образом, насколько это наиболее эффективно с точки зрения цепочки разделения труда. Команда 1 может работать в Москве, команда 2 в Екатеринбурге, команда 3 в Новосибирске. Инфраструктурная поддержка команд разработки может осуществляться из Санкт-Петербурга. Города представлены в качестве иллюстративного примера. В случае создания продуктов, имеющих меньшую зависимость от локализованных правил регулятора, возможно размещение команд в самых разных уголках Земли. Иллюстративно это представлено на Рисунке 14.

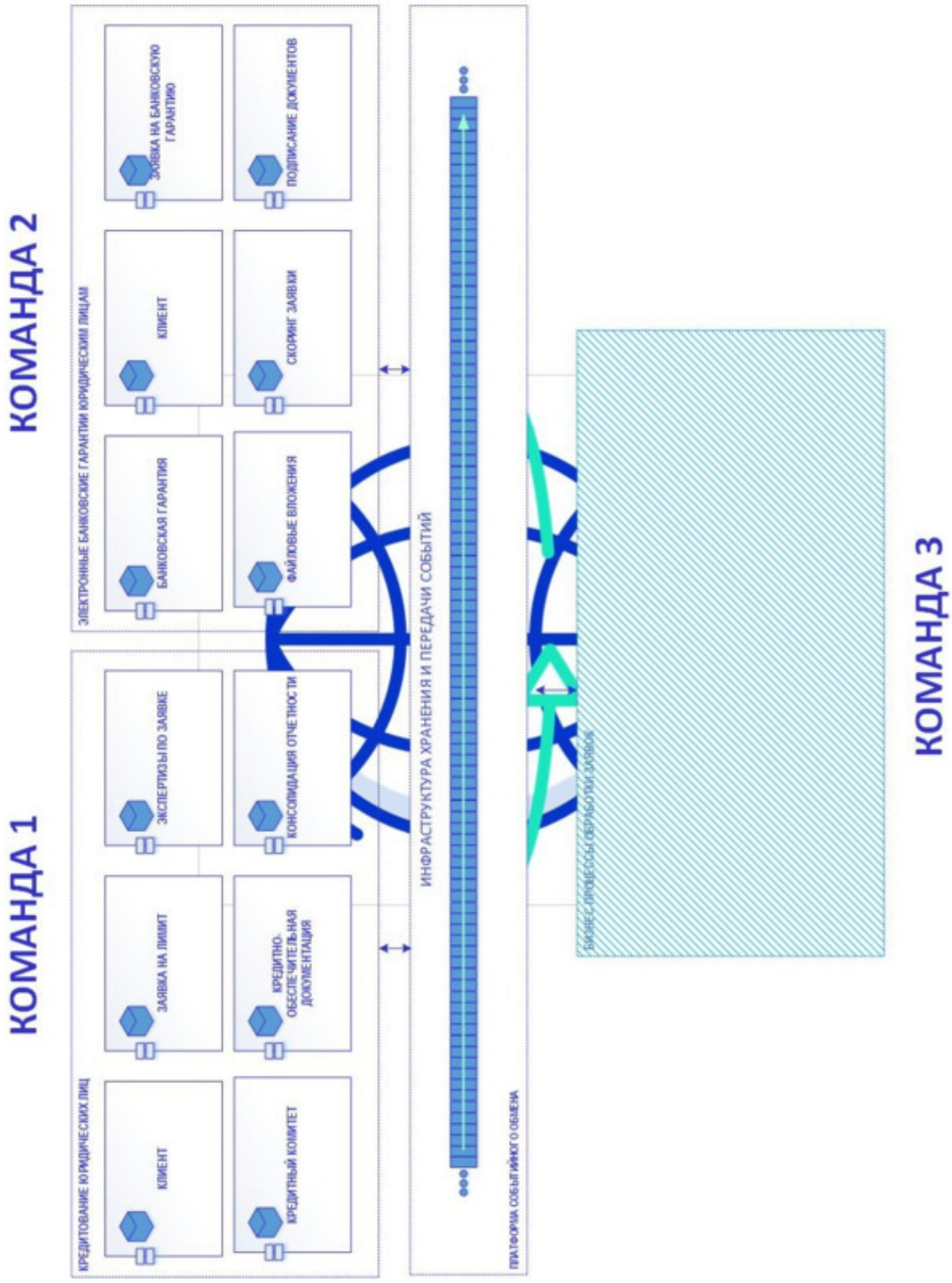

Рисунок 14. Географически распределенная команда разработки

Необходимо учитывать, что при слабой связности компонентов (микросервисов) продуктовых областей между собой их также можно реализовывать распределенной командой. Но данный вариант является на сегодняшний день не слишком реалистичным ввиду сложностей координации продуктовой команды.

Рассмотрев развитие архитектуры в современных условиях, требующих предоставления возможности разработки программного обеспечения распределенными командами, перейдем к рассмотрению второй смысловой плоскости распределенности как одной из ключевых тенденций развития архитектуры – требования к распределенному выполнению создаваемых ИТ-решений.

Как мы уже говорили выше, проводя обзор ключевых тенденций развития архитектуры, требования к распределенному функционированию современных ИТ-решений стали новыми с точки зрения рынка ИТ, что привело к существенному пересмотру подходов к проектированию. Рука об руку с распределенностью пришли требования по приоритетному развитию дистанционных каналов обслуживания клиентов и партнеров. Все эти требования возникли вследствие стремительного роста стартапов и формирования на их основе технологических гигантов. В фильме Дэвида Финчера «Социальная сеть» (The Social Network, 2010) режиссер вкладывает в уста героя Джастина Тимберлейка (играет Шона Паркера) пророческие слова: «Мы жили в деревнях, а потом в городах, а теперь будем жить в Интернете!» Безусловно, и сегодня можно встретить на просторах Интернета ехидные комментарии в адрес данных слов, утверждающие, что жизнь не ушла не только из городов, но и из деревень. Но речь шла не о физическом месте проживания (мы уже видели по ходу настоящего раздела, насколько иллюзорным может становиться его значимость в современном мире), а о mindset. Специально расшифровывая подобные аспекты, персонаж Рашиды Джонс (играет адвоката Мэрилин Делпи) замечает относительно проникновения современных технологий в нашу жизнь: «Босния… У них нет дорог, но есть Facebook». Разумеется, речь идет не о физическом исчезновении деревень и городов, отсутствии необходимости в инфраструктуре обеспечения жизнедеятельности, а о коренном изменении мышления, формировании совершенно нового mindset. Технологические решения проникают в нашу жизнь, причем доступны они в любой точке земного шара. Современный человек не мыслит свою жизнь без Интернета, без сервисов, предоставляемых самыми различными компаниями как в региональном, так и в глобальном масштабе. Но любая медаль имеет обратную сторону – технологические решения, прорвавшись в жизнь человека, связали себя новыми требованиями – они должны быть доступны из любой географической точки и предоставлять неизменно высокое качество сервиса. В противном случае они станут неконкурентоспособными. Подобные требования являются исключительно важными с точки зрения архитектуры.

Ранее по ходу настоящего раздела мы рассмотрели современные организационные, технические и архитектурные практики, применяющиеся для разработки современных ИТ-решений распределенными командами. Как же будут функционировать создаваемые таким образом решения? Например, решение, иллюстративно представленное на Рисунках 13 и 14, должно быть доступно всем юридическим лицам России (кроме, разумеется, тех, на кого наложены предписанные законом ограничения). Рассмотрим функционирование распределенного решения, принимая во внимание те архитектурные принципы, которые уже были предложены для его организации: микросервисная архитектура, продуктовый подход, практики EDA.

Описанные ранее ключевые принципы микросервисной архитектуры имеют ряд следствий практического применения. Одним их них является то, что абсолютное большинство микросервисов проектируется и разрабатывается в формате stateless компонентов, то есть они не сохраняют свое состояние между вызовами – для выполнения данной функции служит внешнее по отношению к микросервису хранилище информации. Таким хранилищем может быть база данных, in-memory data grid (IMDG), платформа событийного обмена (и такие варианты реализации используются, например, The New York Times). С точки зрения самих микросервисов отсутствие сохранения состояния между вызовами позволяет создавать количество экземпляров микросервиса, необходимое для корректной обработки запросов, учитывая возможный рост числа последних. Отсутствие необходимости репликации сессий

на уровне экземпляров микросервисов позволяет минимизировать внимание, уделяемое данному вопросу, и масштабировать микросервисы в допустимых инфраструктурой пределах. При этом крайне важным оказывается наличие располагаемых инфраструктурных мощностей для развертывания соответствующих программных компонентов в таких местах, где сетевая латентность не станет преградой для высокого качества сервиса. Например, финансовая организация, предоставляющая услуги в части продуктов по всей территории России, может быть заинтересована в нескольких центрах обработки данных, которые будут географически распределены.

Если функционирование прикладной логики, реализующей соответствующие продукты, достаточно хорошо ложится на распределенную модель, то возникают вопросы, на какой же уровень переносится растущая сложность исполнения ИТ-решений. Выше уже отмечалось, что для сохранения состояния решений используются внешние по отношению к микросервисам хранилища данных. И вопрос функционирования данных хранилищ в распределенной конфигурации становится исключительно важным. Традиционные решения с централизованными базами данных оказываются слабо применимыми в современных условиях – несколько мощных вычислительных узлов попросту не доставят данные микросервисам за приемлемые временные промежутки. В случае, если речь идет о доступности ИТ-решений по дистанционным каналам, когда время отклика и предоставления услуг (по крайней мере, их части) должно составлять несколько секунд, таковые задержки недопустимы. Создание территориально распределенных кластеров традиционных решений по хранению данных также оказывается проблематичным – используемые такими решениями методы синхронизации и поддержания целостности данных зачастую оказываются несостоятельными в условиях значительной сетевой латентности. Соответственно, мир оказывается заинтересован в принципиально новых хранилищах информации. И такие хранилища, опять же, пришли из стартапов. Например, одно из самых популярных на сегодняшний день решений по построению распределенных баз данных Apache Cassandra было создано в Meta Platforms (ранее Facebook). Аналогично распределенные конфигурации предлагают IMDG решения, такие как Apache Ignite. Отметим, что приводимые примеры современных технологий являются решениями с открытым исходным кодом.

Современные распределенные платформы предполагают совершенно новый уровень производительности и надежности в распределенной среде. Безусловно, модель работы с информацией в данном случае отличается от моделей, принятых в соответствии с традиционными подходами предыдущих архитектурных поколений, синхронизация больших объемов данных в распределенной среде вносит свои ограничения. И здесь на помощь командам разработки приходит большое количество потенциальных топологий рассматриваемых решений. Благодаря глобальной цепочке разделения труда, актуальной для создания подобных технологических решений, становится возможным создать набор потенциальных конфигураций, количественный и качественный состав которого значительно превосходит то, что предлагалось традиционным «закрытым» программным обеспечением. Подобные технологические решения предоставляют новые возможности для проектирования программного обеспечения, но и предъявляют дополнительные требования к работе архитектора. Если ранее он мог ссылаться на известные топологии и лучшие практики (которых было достаточно ограниченное число) «закрытого» программного обеспечения, использовавшегося организацией, то теперь он должен ориентироваться в широком спектре современного открытого программного обеспечения, его возможных топологиях, а также лучших практиках применения последних. Создаваемые информационные системы должны подвергаться тщательному архитектурному анализу на предмет вариантов развертывания, доступа клиентов, сценариев использования, интеграционной составляющей. Результатом анализа станет выбор необходимого программ-

ного обеспечения для реализации, рекомендации по его использованию, направляющие по развитию для команд разработки.

Все сказанное касается не только хранения информации и использования распределенных СУБД. Платформа событийного обмена также должна обеспечивать возможность исполнения решений в распределенной конфигурации. Из современных решений первым кандидатом на подобную роль может считаться Apache Kafka, уже использующаяся в таких компаниях, как вышеупомянутый The New York Times, Linkedin, Uber, «Сбербанк России» и многих других. Решение изначально поддерживает распределенную топологию развертывания и предоставляет широкий спектр возможностей для «тонкой» настройки.

Поскольку микросервисы не сохраняют состояние между вызовами, они нуждаются в предварительной выборке данных для отображения пользователю необходимой ему информации, а также реакции на его запросы. Для обеспечения эффективного доступа к часто используемым данным обычно применяются технологии кэширования (IMDG), также поддерживающие распределенные топологии. Примерами могут служить упомянутый выше Apache Ignite или Infinispan.

Пример решения, представленный на Рисунках 13 и 14, не является исчерпывающим с точки зрения распределенного функционирования современных ИТ-решений. Например, для финансовой сферы актуально выполнение групповых операций, предполагающих проведение однотипных действий над огромными объемами данных. Примером такой операции может служить массовое начисление процентов по счетам. Современные технологии позволяют осуществлять выполнение подобных ресурсоемких операций в оперативной памяти на распределенных узлах обработки, при этом мощность каждого отдельного узла соответствует скорее продвинутому персональному компьютеру, а не гигантскому серверу, что представляет собой разительное отличие от традиционных подходов, стяжавших недобрую славу в профессиональном сообществе и по праву получивших наименование жаргонного типа «залить железом». Аналогичным образом данные могут храниться не только в распределенных базах данных, но и в распределенной файловой системе, что принципиально снижает стоимость хранения (крайне актуально при современном росте объемов хранимых данных).

Таким образом, мы видим, что на всех уровнях создания и развития программного обеспечения востребована концепция распределенного исполнения, под нее создается соответствующий технологический базис. Задача архитектора при этом – не только распределить границы областей разработки решения по продуктовым областям, но и предложить технологические платформы и их топологии, позволяющие максимально эффективно выполнять задачи ИТ-решения по мере реализации последнего. Одновременно учитывается возможность непредсказуемой нагрузки на решение посредством дистанционных каналов. Например, если финансовая организация предложит исключительно выгодный по меркам рынка продукт, каналом предложения которого и соответствующего приема заявок станет сайт компании (без требования регистрации, что выгодно отличает его перед, например, системами дистанционного банковского обслуживания, ДБО), то нагрузка на ИТ-ландшафт может резко возрасти за краткий промежуток времени. Создаваемая архитектура должна позволить решению сохранить корректность функционирования и качество предоставления услуг и при столь агрессивно возрастающей нагрузке.

Основываясь на вышеизложенном, можем отметить, что роль архитектора в рамках ответа на те вызовы, которые ставит перед архитектурой фактор распределенности, существенно изменилась. На сегодня необходимо обладать знанием широкого спектра программного обеспечения, рассчитанного на функционирование в распределенной гетерогенной среде, знать его основные топологии и сценарии их применения, погружаться в предметную область для качественной грануляции проектируемых информационных систем, позволяющей вести разработку силами распределенных команд. На сегодня архитектор – это не выделенный спе-

циалист, привлекаемый командами «по случаю», но лидер технологических изменений. Он задает ключевые направляющие создания, развития решений, а также их технологического наполнения.

Архитектура и бизнес-процессы

Важнейшей тенденцией развития архитектуры является ее способность обеспечить качественное и эффективное исполнение бизнес-процессов, актуальных для организации, автоматизация деятельности которой рассматривается.

Любая современная организация обеспечивает структуризацию своей деятельности на основании ее упорядочивания с последующим описанием и формирования бизнес-процессов, которые в удобочитаемом формате (например, в BPMN-нотации) обеспечивают наглядное и доступное описание выполняемых действий. При этом к бизнес-процессам обычно предъявляются требования по обеспечению возможности моделирования, определения этапа исполнения, а также задания количественных метрик (KPI), позволяющих проводить их анализ и оптимизацию.

Для обеспечения автоматизированного исполнения бизнес-процессов используются движки исполнения бизнес-процессов (BPM-движки), обеспечивающие следующие основные возможности:

• Моделирование процесса при помощи наглядных визуальных средств (полнота поддержки BPMN-нотации может варьироваться в зависимости от конкретного моделирующего средства).

• Задание связей, как выполняемых в рамках процесса действий, так и интеграций с внешними системами, которые ответственны как за непосредственное выполнение операций, так и могут предоставлять интерфейсы автоматизированных рабочих мест для выполнения задач пользователями (внутренними и внешними).

• Автоматизированное выполнение шагов процесса в соответствии с заданным алгоритмом.

• Назначение ролей, ответственных за этапы выполнения процесса.

• Определение метрик и KPI процесса и его составляющих, а также ролей, задействованных в процессе.

При этом в течение долгого времени BPM-движки представляли собой сложные платформы автоматизации монолитного характера, которые осуществляли управление всеми бизнес-процессами организации. Указанный подход содержал серьезные риски, которые затрудняли эффективное внедрение средств автоматизации управления бизнес-процессами:

• Невысокая производительность средств автоматизации процессов стала одним из основных препятствий их внедрения. Зачастую многие организации осуществляли автоматизацию бизнес-процессов, жестко программируя их выполнение, при этом основной мотивацией такого выбора была именно забота о производительности.

• Монолитный характер BPM-движков приводил к сложной технологической связи бизнес-процессов (зачастую не столь тесно связанных между собой на уровне потребностей бизнеса) и невозможности их независимого масштабирования, что увеличивало затраты на инфраструктурное обеспечение приемлемого качества функционирования.

• Поддержание большого количества бизнес-процессов на уровне монолитного BPM-движка явным образом ставило перед организациями проблему создания общей модели данных, применимой для всех бизнес-процессов, либо обеспечения согласования различных используемых моделей данных – оба приведенных варианта направления работы влекли серьезные трудозатраты, за которыми логичным образом следовали и финансовые затраты.

Пример монолитного BPM-движка в ИТ-ландшафте схематично показан на Рисунке 15.

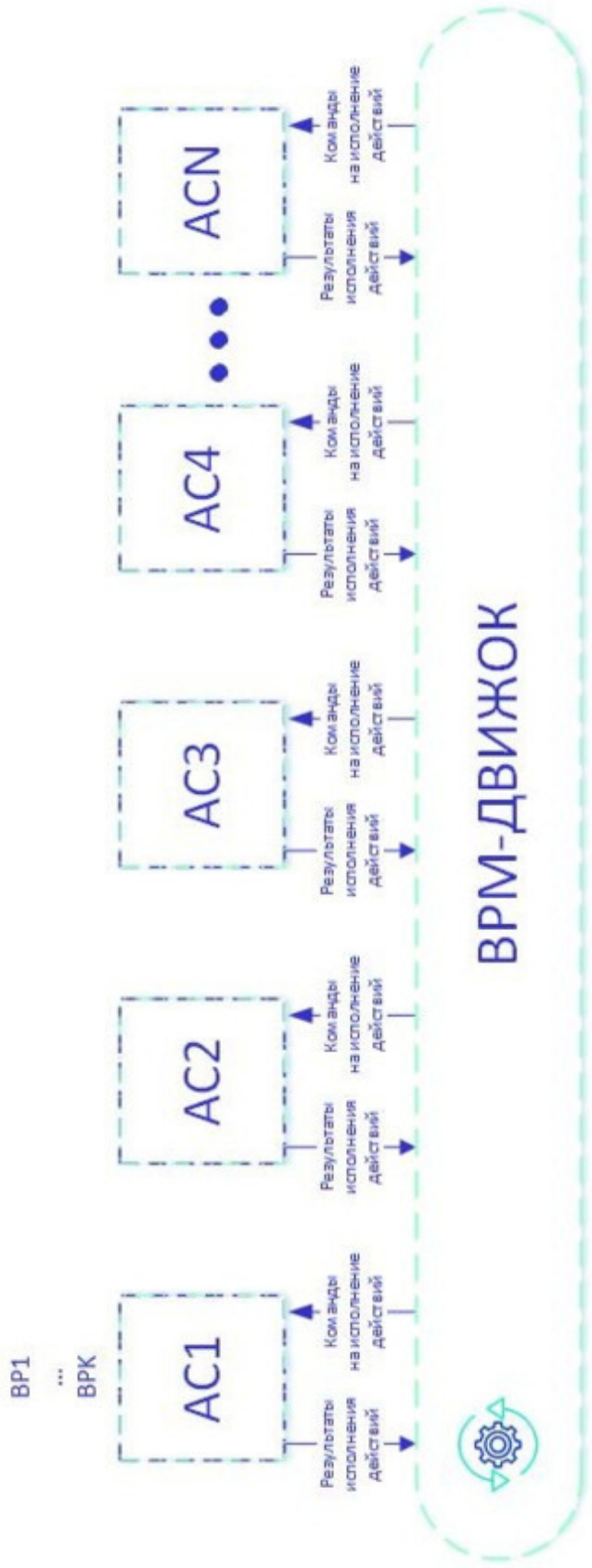

Рисунок 15. Пример монолитного BPM-движка в ИТ-ландшафте

На Рисунке 15 представлен BPM-движок, обеспечивающий выполнение бизнес-процессов ИТ-ландшафта, выстроенного в монолитной или SOA архитектуре. Некоторое число

информационных систем (N – на рисунке) обеспечивают автоматизацию бизнес-действий, организуемых в некоторое число бизнес-процессов (K – на рисунке). Соответственно, монолитный движок должен поддерживать значительное число (в современной организации оба параметра могут варьироваться от сотен до тысяч) сочетаний бизнес-действий и систем. При этом независимое масштабирование невозможно.

Подобная топология развертывания решения по автоматизации бизнес-процессов затрудняла разработку ИТ-решений распределенными командами разработки, а распределенное исполнение (в современном понимании, подробно рассмотренном выше) информационных систем и ИТ-ландшафта в целом исключалось. Долгое время лидирующая роль на рынке BPM-движков принадлежала «закрытым» решениям, поставляемым крупными компаниями – разработчиками программного обеспечения, таким как IBM и Oracle.

Ранее рассматривались такие архитектурные практики трансформации как переход к микросервисной архитектуре, использование продуктового подхода и EDA. Нетрудно заметить, что при реализации новых технологических решений в парадигме микросервисной архитектуры монолитный BPM-движок становится узким местом в части скорости разработки: каждая информационная система в микросервисной парадигме разрабатывается несколькими продуктовыми командами, что в свою очередь предъявляет серьезные требования к развитию BPM-движка. Нельзя не отметить, что резко возросшие требования к производительности создаваемых решений предъявляют качественно иные требования к масштабированию всех компонентов ИТ-ландшафта, в том числе BPM-движку.

В рамках перехода к микросервисной архитектуре и продуктовому подходу, подробно рассмотренным в предыдущих разделах, меняется и концепция управления бизнес-процессами. Продуктовый подход предполагает, что силами команды гибкой разработки (возможно, нескольких команд при сложном многофункциональном продукте) будет осуществляться полный цикл работ по автоматизации продукта, предоставляющего самодостаточную ценность для клиентов и/или партнеров. Такой продукт подразумевает в том числе автоматизацию сложных бизнес-процессов, связанных с обработкой его жизненного цикла. При этом отдельные элементы таких бизнес-процессов могут быть действиями, реализуемыми на уровне распределенных компонентов (микросервисов), но их объединение является действием иного порядка, которое имеет смысл назвать управлением бизнес-процессом с продуктовой спецификой или продуктовым бизнес-процессом, в соответствии с современными архитектурными практиками также являющимся микросервисом. Иллюстративно данная концепция представлена на Рисунке 16.

Для иллюстрации взят пример продукта – электронной банковской гарантии – и один бизнес-процесс – заведение банковской гарантии. В соответствии с ранее представленными подходами к проектированию, управление бизнес-процессом реализовано в формате микросервиса. Отметим, что бизнес-процесс как управляющий компонент (в отличие от компонентов, непосредственно выполняющих бизнес-действия), вынесен за пределы продуктовой области и взаимодействует с продуктовыми микросервисами посредством платформы событийного обмена. Структура автоматизации продуктовых бизнес-процессов выглядит следующим образом:

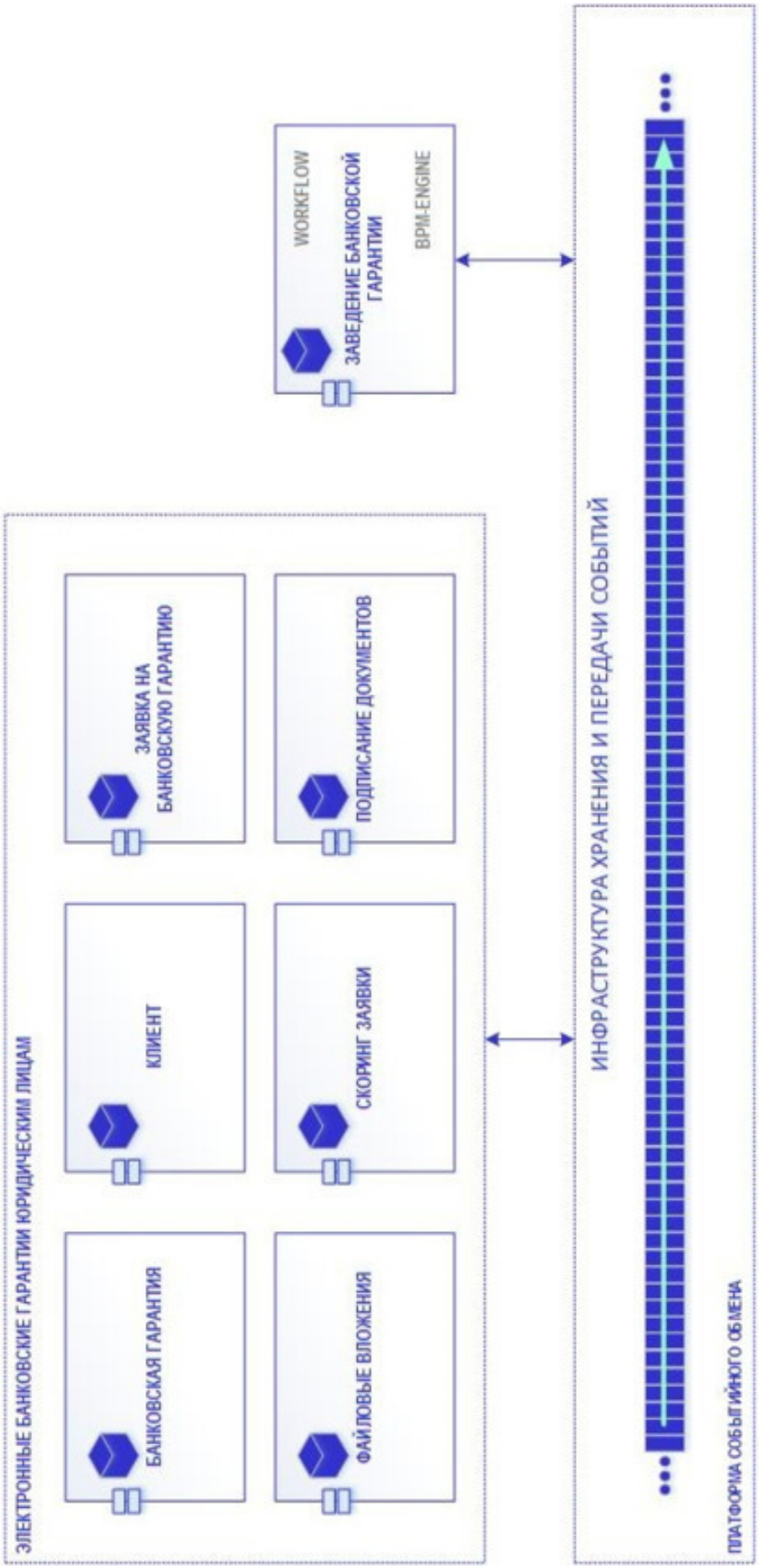
ЭЛЕКТРОННЫЕ БАНКОВСКИЕ ГАРАНТИИ ЮРИДИЧЕСКИМ ЛИЦАМ
БАНКОВСКАЯ ГАРАНТИЯ
ФАЙЛОВЫЕ ВЛОЖЕНИЯ
КЛИЕНТ
СКОРИНГ ЗАЯВКИ
ЗАЯВКА НА БАНКОВСКУЮ ГАРАНТИЮ
ПОДПИСАНИЕ ДОКУМЕНТОВ
WORKFLOW
ЗАВЕДЕНИЕ БАНКОВСКОЙ ГАРАНТИИ
BPM-ENGINE
ИНФРАСТРУКТУРА ХРАНЕНИЯ И ПЕРЕДАЧИ СОБЫТИЙ
ПЛАТФОРМА СОБЫТИЙНОГО ОБМЕНА

Рисунок 16. BPM-движок в микросервисной архитектуре

• Каждый набор микросервисов, обеспечивающих автоматизацию жизненного цикла продукта, должен иметь возможность осуществлять управление бизнес-процессами (содержать один или более BPM-движков), ассоциированными с данным продуктом или группой продуктов.

• Применяемые технологии BPM-движков должны обеспечивать простоту и удобство использования в микросервисной топологии развертывания.

• BPM-движок должен либо обладать высокой производительностью, либо предоставлять широкие возможности масштабирования (рекомендуемым является сочетание возможностей).

• Хранение описаний (схем) бизнес-процессов, исполняемых продуктовым BPM-движком, должно быть локальным по отношению к автоматизации бизнес-функционала. Схемы не должны запрашиваться из удаленного реестра, что позволяет избежать в том числе проблем сетевой латентности.

Важным требованием к системе управления бизнес-процессами в современной архитектуре с учетом применения продуктового подхода и практик EDA является также следующее: низкий порог входа в том числе с точки зрения компетенций. Из представленных на сегодняшний день на рынке решений, удовлетворяющих всем перечисленных требованиям, можно отметить BPM Camunda, являющееся программным обеспечением с открытым исходным кодом.

Как отмечалось выше, реальные бизнес-процессы, выполняющиеся в современных организациях, обладают сложной структурой, включающей сотни и тысячи шагов и бизнес-действий. В случае выполнения подобных процессов одним микросервисом последний нарушит одно из ключевых требований, предъявляемых в парадигме микросервисной архитектуры, заключающееся в простоте каждого микросервиса. Рассмотрим, каким образом возможно решить таковую проблему.

Решение проблемы переноса управления сложными бизнес-процессами в парадигму микросервисной архитектуры состоит из нескольких этапов. Первым из них является технический рефакторинг бизнес-процессов.

Важным свойством, облегчающим применение BPM-движков, является наличие развитых средств моделирования процессов с использованием общепринятых нотаций (стандартом де-факто является BPMN). Использование подобных средств качественно улучшает возможности взаимодействия владельца продукта, оперирующего бизнес-понятиями, и команды гибкой разработки, состоящей из технических специалистов.

Вместе с тем, подобный подход несет в себе некоторые риски:

• Отражение в бизнес-процессе исключительно логики бизнес-уровня – необходимой, но не достаточной части автоматизации, предполагающей исполнение и технических задач. В данном случае расширение техническими задачами специализированных бизнес-действий (компонентов бизнес-процесса) может оказаться некорректным с точки зрения потенциального масштабирования создаваемого ИТ-решения. Более корректным стоит признать включение в состав бизнес-процесса в том формате, в котором уже производится автоматизация, отдельных этапов технического характера. Это и становится первой стадией технического рефакторинга бизнес-процесса.

• Отрисовка инструментальными средствами бизнес-процесса целиком без предварительного мелкого гранулирования. В рассматриваемом варианте существует серьезный риск создания монолитного артефакта – бизнес-процесса, скорость разработки которого и внесения изменений будет существенно ниже, нежели на уровне продуктовой логики. С учетом того, что изменения в продуктовых бизнес-процессах могут инициироваться на регулярной основе

(в зависимости от потребностей рынка, участником которого является автоматизируемая организация), озвученный риск следует признать исключительно важным. Соответственно, приступая к автоматизации бизнес-процесса, следует всегда рассматривать возможность его декомпозиции на составляющие, что становится вторым этапом технического рефакторинга бизнес-процесса.

Таким образом, при осуществлении технического рефакторинга выполняются следующие основные задачи:

• Формирование технической карты бизнес-процесса с учетом детализированного бэклога продукта, содержащего как пользовательские истории, так и технические задачи, выполнение которых необходимо в рамках реализации продуктовой логики.

• Анализ компонентов бизнес-процесса на предмет повторяемости и независимости с выделением в отдельные исполняемые единицы.

• Анализ контекста процесса на предмет разрыва и возможности разделения на составляющие по границам контекста.

Понятие контекста бизнес-процесса было сформулировано еще в период внедрения BPM-систем в классической архитектуре монолитного типа. Данный термин актуален и в настоящее время. Под контекстом бизнес-процесса понимается совокупность информации, характеризующая выполняемые в ходе бизнес-процесса действия, получаемые и обрабатываемые данные, а также служебная информация, необходимая для обработки бизнес-данных.

На основании понятия бизнес-процесса вводится понятие бизнес-транзакции как набора связанных действий, выполняемых в рамках бизнес-процесса. При этом выделяются локальные транзакции, которые могут выполняться в рамках связанного неделимого набора действий (в качестве примера такой локальной транзакции можно рассматривать процесс скоринга кредитной заявки), могущего исполниться лишь целиком и подлежащего отмене в случае возникновения ошибок, и глобальные, состоящие из последовательности локальных. Глобальная транзакция не может быть отменена, поскольку зафиксированы результаты исполнения локальных транзакций, входящих в ее состав. В случае возникновения ошибок по ходу выполнения глобальной транзакции отмена результатов локальных транзакций, завершившихся успешно, возможна посредством выполнения компенсационных действий (алгоритмов), которые, в свою очередь, также могут быть сложными бизнес-процессами. Примером невозможности отмены является ошибочное уведомление клиента о тех или иных событиях по ходу исполнения бизнес-процесса. В данном случае компенсационным алгоритмом может быть отправка корректирующих уведомлений.

Каждая локальная транзакция характеризуется своим транзакционным контекстом, определяющим ее состояние. В общем случае контексты локальных транзакций недоступны друг другу. Контекст бизнес-процесса, представляющего собой глобальную транзакцию, определяется набором контекстов локальных транзакций, входящих в его состав. При этом локальные транзакционные контексты являются непрозрачными друг для друга и могут управляться независимо посредством BPM-движка (в парадигме микросервисной архитектуры – различных движков).

Задачи, требующие неавтоматизированного вмешательства пользователя или вызова внешней (по отношению к управляющему бизнес-процессом BPM-движку) системы, могут выполняться непредсказуемо долго, в результате чего их реализация должна осуществляться в рамках отдельных локальных транзакций с собственным контекстом. Совокупность локальных транзакций и их контекстов составляет общую глобальную транзакцию бизнес-процесса и ее контекст соответственно. Аналогично в отдельном контексте должен выполняться вложенный бизнес-процесс.

Важным аспектом технического рефакторинга и управления контекстом бизнес-процесса является то, что реальное значение для бизнеса имеет глобальная транзакция и ее контекст, соответственно, при обеспечении рефакторинга необходимо поддерживать целостность глобального контекста бизнес-процесса. Этой цели служат такие шаблоны управления исполнением бизнес-процессом, как оркестровка и хореография.

В ходе внедрения систем управления бизнес-процессами (еще в парадигме монолитных решений и SOA) был определен термин оркестровка. Оркестровка – централизованное управление бизнес-процессом, осуществляемое из одной точки. К преимуществам оркестровки относят удобство осуществления мониторинга бизнес-процесса, вся информация о состоянии и исполнении которого содержится в одном компоненте ИТ-ландшафта организации. К недостаткам – сложность и перегруженность этой одной точки управления исполнением (выше уже говорилось о том, что исполняющий таким образом реальный бизнес-процесс микросервис теряет соответствие требованию простоты реализации).

В противоположность оркестровке вводится шаблон хореографии. Хореография предполагает отсутствие единой точки управления исполнением бизнес-процесса – компоненты, которые должны действовать согласованно, публикуют события в соответствии с практиками EDA, которые прослушиваются другими компонентами, составляющими в совокупности бизнес-процесс. К достоинствам данного шаблона проектирования можно отнести практически неограниченные возможности масштабирования под высокой нагрузкой, соответствующей современным дистанционным каналам. К недостаткам – сложные алгоритмы организации сквозного мониторинга бизнес-процессов.

Как видно из приведенного описания, каждый способ организации управления бизнес-процессами имеет свои преимущества и недостатки, поэтому в современных архитектурных практиках указанные шаблоны скомбинированы следующим образом:

• Для каждого бизнес-процесса, подлежащего автоматизации, проводится технический рефакторинг в соответствии с концепцией, изложенной выше: составляется техническая карта, включающая как бизнес-действия, так и необходимые технические задачи, формируется контекст процесса, выделяются составляющие его локальные контексты.

• Бизнес-процесс, выполняющийся в рамках одного локального контекста, управляется в соответствии с шаблоном проектирования оркестровки, что обеспечивает его максимальную прозрачность с точки зрения мониторинга.

• Оркестровка локального бизнес-процесса осуществляется с использованием BPM-движка в соответствии с лучшими практиками переноса последнего в парадигму микросервисной архитектуры, представленными ранее по ходу настоящего раздела. Примером такого движка может являться BPM Camunda.

• Поскольку оркестровка локального процесса осуществляется посредством выделенного микросервиса, масштабирование производится независимо от других сервисов (как содержащих логику управления локальными бизнес-процессами, так и тех, в которых она отсутствует – микросервисах, осуществляющих исполнение продуктовой логики).

• Управление бизнес-процессами, выполняющимися в рамках отдельных локальных транзакций, но составляющих общий глобальный бизнес-процесс, осуществляется на основе шаблона проектирования хореографии.

• Применение шаблона проектирования хореографии для управления глобальным процессом предполагает использование лучших современных практик EDA (наличия платформы событийного обмена, носящей технический характер), а также специальной организации мониторинга его состояния и исполнения экземпляров бизнес-процессов. В качестве платформы событийного обмена может использоваться, например, Apache Kafka.

• Каждый экземпляр процесса, выполняющегося в рамках локального контекста, может масштабироваться независимо от экземпляров процессов, исполняющихся в рамках иных локальных контекстов.

Отметим, что представленный подход позволяет обеспечивать в автоматизации управления бизнес-процессами следование принципам распределенности. Локальные бизнес-процессы могут исполняться независимо друг от друга, распределяясь, например, по территориальному признаку. Каждый экземпляр локального бизнес-процесса при этом масштабируется независимо не только от других бизнес-процессов, но и от экземпляров локальных бизнес-процессов, составляющих вместе с ним общий бизнес-процесс с глобальным контекстом. Например, в общем бизнес-процессе может быть нагруженный этап, содержащий большое количество неавтоматизированных пользовательских задач. В результате объекты, идущие по бизнес-процессу, например, заявки на электронные банковские гарантии, скапливаются именно на данном этапе. Распределенная структура бизнес-процесса позволяет масштабировать процессный микросервис (workflow микросервис), отвечающий именно за этот этап, не осуществляя аналогичного масштабирования с другими составляющими бизнес-процесса. Подобный подход приводит к существенной экономии инфраструктурных мощностей. Необходимо также отметить, что workflow микросервис, выполняющий обобщенные задачи, может быть задействованным в качестве локального бизнес-процесса в нескольких глобальных процессах, например, в процессах прохождения кредитных заявок и заявок на электронную банковскую гарантию.

Пример распределенного бизнес-процесса, выполняющегося совместно с продуктовыми микросервисами, представлен на Рисунке 17.

На Рисунке 17 представлена совместная работа микросервисов, автоматизирующих прикладную логику продукта электронных банковских гарантий, и процессных микросервисов, отвечающих за функционирование бизнес-процесса. Проиллюстрирован один бизнес-процесс – заведение электронной банковской гарантии (рассматривается поступление заявки на гарантию с внешней площадки). По итогам технического рефакторинга общий бизнес-процесс, выполняющийся в рамках глобального контекста обработки заявки, разбит на 5 локальных бизнес-процессов, выполняющихся в рамках собственных локальных контекстов. Каждый локальный процесс реализуется в рамках микросервисной парадигмы и осуществляет оркестровку действий, входящих в него и реализуемых прикладными микросервисами, пользовательскими неавтоматизированными задачами или внешними по отношению к рассматриваемому ИТ-решению системами. В рамках глобального процесса для синхронизации исполнения локальных элементов используется шаблон проектирования хореография в соответствии с принципами EDA, основой реализации выступает платформа событийного обмена.

Конец ознакомительного фрагмента.