Глава

5

Фабрики программного обеспечения

Программные продукты в некоторых отношениях подобны вещественным продуктам других инженерных дисциплин — таким как мосты, здания и компьютеры. Однако имеется одно существенное отличие, которое придает производству программ свой, уникальный привкус.

Питер Вегнер (Peter Wegner)

В этой главе мы расскажем об индустриализации в области разработки программных продуктов. Затем представим фабрики программного обеспечения как способ продвижения и поддержки индустриализации. Далее рассмотрим то, как реализованы фабрики программного обеспечения и представим простой пример такой фабрики. Затем мы обратимся к последствиям индустриализации. И, наконец, кратко расскажем о том, что должно быть сделано для реализации этого представления, подготовив основу для части 2.

Индустриализация разработки программного обеспечения

Следующая парадигма индустриализует разработку программного обеспечения.

В предисловии и главе 1 мы предположили, что следующей парадигмой разработки должна стать индустриализация производства программного обеспечения, как следствие движения отрасли к зрелости. Что может означать индустриализация в данной области? Конечно, пока она не произошла, мы не можем знать этого в точности. Однако мы можем выдвигать обоснованные предположения на основе наблюдений за развитием отрасли, произошедшим до сегодняшнего дня. Некоторое представление об этом можно также получить, рассматри-

вая индустриализацию в других областях, и сравнивая их с интересующей нас отраслью, чтобы найти возможные сходства и различия между ними.

По иронии судьбы большинство процессов, происходящих в других отраслях, автоматизированы программными системами.

Ранее мы упоминали, что другие отрасли научились подгонке и сборке стандартных компонентов для производства схожих, но разных продуктов, для стандартизации, интеграции и автоматизации производственных процессов, для разработки расширяемых инструментов и конфигурирования их на выполнение повторяющихся задач, для балансирования отношений между поставщиком и потребителем с целью оптимизации затрат и рисков, для автоматизации производства вариантов продукта на основе линеек продуктов и для распределения производства среди высокоспециализированных и взаимозависимых поставщиков. В автомобильной промышленности, например, легковые автомобили и грузовики являются конечными продуктами, которые собираются из стандартных компонентов, поставляемых огромным числом вышестоящих поставщиков, которые, в свою очередь, собирают эти компоненты из составных частей, полученных от других производителей. Конечно, цепочки поставщиков — не уникальное явление для отраслей производства материальных продуктов. Их можно обнаружить в области финансовых услуг, ипотечного кредитования, музыкальной индустрии, телевидении, кинопроизводстве, издательском деле и многих других отраслях, связанных с информацией и правами собственности. По иронии судьбы большинство производственных процессов в этих областях автоматизированы программными системами. В данном случае оказывается справедливой поговорка “сапожник без сапог”.

Некоторые сравнивают яблоки с апельсинами.

Некоторые проводят аналогии между программной индустрией и отраслях, имеющими дело с физическими товарами. Иногда такие дискуссии похожи на сравнение яблок с апельсинами, однако, в приложении к разработке программ и материальных товаров. Ключ к прояснению этой путаницы лежит в понимании экономических аспектов повторного использования.

Экономика повторного использования

Повышение уровня абстракции требует инвестиций в повторно используемые производственные активы.

Вспомните из главы 2, что повторное использование решений общих подпроблем в данном домене может снизить общий уровень затрат на решение множества проблем в домене. Повторное использование также сокращает время выхода на рынок и повышает качество продуктов [127]. Чтобы обеспечить возврат инвестиций в разработку решений, нужно обеспечить столько случаев повторного использования, чтобы покрыть стоимость их разработки — либо непосредственно, через снижение затрат, либо косвенно, через сокращение времени выхода на рынок и повышение качества. Другими словами, мы можем думать об этих решениях с точки зрения инвестиций или с точки зрения производственных активов. Мы уже видели некоторые производственные активы, такие как языки, инструменты, шаблоны и каркасы. Эти производственные активы сопровождаются процессными активами, поддерживающими их применение, среди которых сценарии и среды тестирования, дизайн или пользовательская документация. В конечном итоге процессные активы поддерживаются инструментами и другими средствами автоматизации. Теперь мы можем определить производственный актив как про-

граммный артефакт, предназначенный непосредственно для возврата инвестиций за счет повторного использования.

Вспомните из главы 4, что подход к повторному использованию "от случая к случаю" дает несущественные результаты, и часто приводит к разочарованию и скептицизму в отношении идеи повторного использования кода, разработанного кем-то другим. Чтобы получить преимущества от повторного использования, описанные ранее, следует применить более зрелый подход, предполагающий идентификацию общих подпроблем в заданном домене и разработку интегрированных коллекций производственных активов, которые могут быть повторно использованы для решения этих проблем предсказуемым образом. Цель этого подхода, называемого систематическим повторным применением, состоит в повышении уровня возврата инвестиций в производственные активы через экономию за счет масштаба и области применения.

Чтобы быть экономически эффективным, повторное использование должны быть систематическим, а не от случая к случаю.

Экономия за счет масштаба и области применения

Экономию за счет масштаба часто путают с экономией, связанной с областью применения. И то, и другое сокращает время и затраты при повышении качества за счет массового, а не индивидуального производства большего количества продуктов, однако между ними есть отличия. Чтобы устранить эту путаницу, мы определим оба термина и рассмотрим, как и когда проявляются оба феномена.

Экономия за счет масштаба и области применения отличаются.

Экономия масштаба возникает, когда множество идентичных экземпляров одного дизайна производятся массово, а не индивидуально, как показано на рис. 5.1. Такую экономию обеспечивает производство продуктов вроде метизов, когда производственные активы, такие как металлообрабатывающие станки, используются для выпуска множества идентичных экземпляров продукции. Вначале, в рамках ресурсоемкого процесса, называемого разработкой и выполняемого инженерами, создается дизайн вместе с несколькими начальными экземплярами — прототипами. Множество дополнительных экземпляров, называемых копиями, обычно создаются в рамках другого процесса, называемого производством и выполняемого машинами с применением низко квалифицированного труда, чтобы удовлетворить потребность рынка в невысокой цене. Мы можем воспринимать экономию масштаба как снижение стоимости многократного решения одной и той же проблемы одним и тем же способом, за счет снижения стоимости производства каждого решения.

Экономия масштаба возникает, когда производственные активы используются повторно для массового производства копий прототипа.

Экономия масштаба возникает, когда множество подобных, но отдельных дизайнов и прототипов производятся в массовом масштабе вместо индивидуального, как показано на рис. 5.2. Например, в автомобильном производстве множество подобных, но отдельных дизайнов продукта часто разрабатывается путем комбинирования существующих проектных решений, таких как шасси, кузов, интерьер и система передач, а вариан-

Экономия масштаба возникает, когда производственные активы повторно используются для разработки множества подобных, но отдельных дизайнов и прототипов.

ты и модели часто разрабатываются на основе вариаций узлов, подобных двигателю и подвеске, из существующего дизайна. Другими словами, одни и те же приемы, процессы, инструменты и материалы используются для проектирования и прототипирования множества похожих, но разных продуктов. Обратите внимание, как это напрямую соответствует концепции линеек программных продуктов, представленной в главе 4. То же верно и для коммерческого строительства, когда множество мостов и небоскребов редко имеют совершенно одинаковый дизайн. Интересно, что в коммерческом строительстве на основе каждого удачного проекта создается один–два экземпляра сооружения, так что там экономия масштаба реализуется нечасто.

По этой причине в коммерческом строительстве на первый план выходит экономия за счет области применения. В автомобильной промышленности, где обычно производится множество идентичных экземпляров на основе каждого удачного проекта, экономия масштаба обычно дополняется экономией области применения, как показано на иллюстрации.

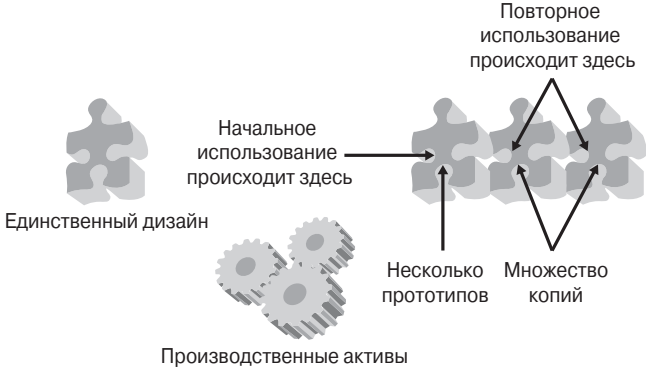

Рис. 5.1. Экономия масштаба

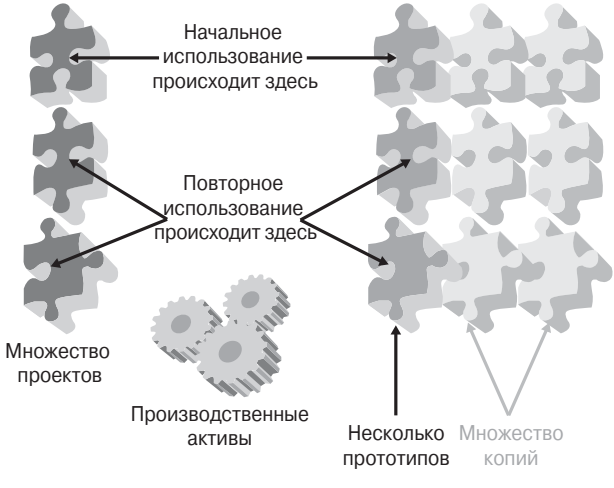

Рис. 5.2. *Экономия области применения*

Мы можем думать об экономии области применения, как о снижения стоимости решения множества похожих, но разных проблем в заданном домене посредством массового решения их часто встречающихся подпроблем с последующей сборкой, адаптацией и конфигурированием результирующих частичных решений для решения проблем верхнего уровня.

Систематическое повторное использование программного обеспечения

Программное обеспечение на некоторых рынках может обеспечить экономию масштаба за счет массового производства посредством дублирования.

Программное обеспечение подобно производству автомобилей на одних рынках и коммерческому строительству — на других. На рынках, где массово производятся копии, такие как пользовательские настольные программы вроде текстовых процессоров, обеспечивается экономия масштаба, как в автомобильной промышленности. На рынках, подобных производству автомобилей, разработка и производства — совершенно разные процессы. В ресурсоемком процессе, который осуществляют инженеры и который называется “разработкой”, создается дизайн вместе с начальным экземпляром, называемым эталоном. Прочие экземпляры, называемые копиями, производятся в другом процессе, называемом дублированием и выполняющимся с помощью машин и низко квалифицированной работы, что обеспечивает потребность рынка в низкой цене продукта.

Экономия масштаба теряется, когда требуется высокий уровень настройки и подгонки.

Конечно, есть существенные отличия между этими двумя отраслями, в числе которых стоимость производства и потребления копий. В автомобильной промышленности, как и в других отраслях тяжелой промышленности, производство копий требует дорогостоящих ресурсов, в то время как в программной индустрии — только дешевых цифровых и бумажных носителей. В этом смысле программная индустрия подобна индустрии развлечений, где легкость и дешевизна производства и распространения копий приводит к проблемам с лицензированием и защитой прав собственности. Другое различие между производством программ и автомобилей заключается в том, что программы обычно дороже потреблять из-за настройки и подгонки. Настройка программ может простираться от установки предпочтений до программирования, тогда как автомобили обычно требуют совсем незначительной настройки. В случае достаточно объемной и сложной настройки, которой требуют пакетные приложения промышленного масштаба, копии получаются похожими, но разными продуктами — при этом экономия масштаба исчезает.

Программное обеспечение обеспечивает экономию за счет области применения.

На некоторых рынках, подобных рынку приложений масштаба предприятия, когда бизнес-приложения предназначены для обеспечения преимуществ в конкуренции, а копии производятся достаточно редко, программное обеспечение редко обеспечивает экономию за счет масштаба. На этих рынках оно дает экономию за счет области применения — как и в коммерческом строительстве. Экономия области применения обеспечивается, когда размер рынка ограничен, либо когда перед эксплуатацией требуется дорогостоящая настройка. Такая экономия возникает, когда одни и те же приемы, процессы, инструменты и материалы используются для разработки множества схожих, но разных дизайнов и их первых экземпляров. Поскольку экономия за счет масштаба редко возникает на таких рынках, эко-

номия от области применения, прежде всего, означает необходимость в систематическом повторном использовании.

Экономия от области применения возникает при разработке, в то время как экономия от масштаба — при производстве.

Теперь мы можем видеть, где именно яблоки сравниваются с апельсинами в дебатах об индустриализации программного обеспечения. Производственный процесс в отраслях, имеющих дело с физической продукцией, наивно сравнивается с процессом разработки программного обеспечения. Другими словами, мы не можем сравнивать программную индустрию с индустрией, выпускающей физические товары, оценивая экономию от масштаба, сопровождающую производство физических товаров, в процессе разработки программного обеспечения. Мы можем, однако, ожидать от индустриализации разработки программ экономии области применения, особенно на рынках с ограниченными поставками. Вегнер отмечает, что поскольку программное обеспечение — сущность логическая, а не физическая, затраты в основном сконцентрированы на разработке, а не на производстве (то есть стоимость производства копий готового программного обеспечения незначительна), и поскольку программное обеспечение не изнашивается, его надежность зависит от логических свойств, таких как корректность и устойчивость, а не от физических свойств вроде твердости или податливости.

Индустриализация обеспечит экономию области применения.

Поскольку экономия масштаба возникает во время производства, в то время как экономия области применения — во время разработки, некоторые наблюдатели ошибочно заключают, что производство программного обеспечения фундаментально отличается от отраслей, имеющих дело с физическими продуктами. Это заключение неверно, потому что создает впечатление о программной индустрии, как о чем-то особенном, отвергая критику со стороны других отраслей и финансового сообщества, которые требуют, чтобы производство ПО, как и другие отрасли, приносило совокупный положительный возврат инвестиций, а не наоборот, и в конечном итоге должно оцениваться потенциальными инвесторами исключительно с точки зрения экономической выгоды [36]. Безусловно, экономические соображения в конечном итоге превалируют. Программная индустрия в основе своей похожа на другие отрасли — инженеры разрабатывают проекты и прототипы, а копии производятся механически, чтобы удовлетворить массовый спрос. Можно надеяться, что программная индустрия, наконец, обратится в лоно индустриализации, прежде чем финансовое сообщество перестанет принимать извинения и заберет свои деньги. Фабрики программного обеспечения и цепочки поставщиков — вот ключевые шаги на пути индустриализации.

Интеграция важнейших нововведений

Фабрики программного обеспечения интегрируют важнейшие нововведения в форме высоко автоматизированного подхода к разработке программ.

Мы говорили в главе 1, что фабрики программного обеспечения интегрируют критические инновации для определения высоко автоматизированного подхода к разработке программ, обеспечивающего экономию области применения. Затем в главе 4 мы идентифицировали эти критические инновации, и показали, как они решают хронические проблемы, которые не в состоянии преодолеть объектная ориентация. Подводя итог этой дискуссии, опишем эти критические инновации.

- Построение семейств сходных, но разных программных продуктов для обеспечения более систематического подхода к повторному использованию.
- Сборка самоописываемых служебных компонентов, используя новые технологии инкапсуляции, пакетирования и оркестровки.
- Разработка специфичных для домена языков и инструментов, используя новые определения языка, генерацию кода и технологии инструментальной разработки, сокращающие объем жесткого кодирования.
- Использование ограничивающего планирования и активного руководства в контексте процессного каркаса для масштабирования к более крупным проектам, географической распределенности и расширенному жизненному циклу продукта, без потери подвижности.

Хотя каждая из этих технологий достаточно зрелая, чтобы быть развернутой в промышленных масштабах, их интеграция создает целое, которое больше, чем сумма его частей.

Фабрики программного обеспечения применяют эти важнейшие нововведения к знакомому 4-этапному шаблону автоматизации разработки, значительно удешевляя реализацию.

В предисловии и в главе 4 мы предположили, что фабрики программного обеспечения применят эти критические инновации к шаблону, состоящему из 4-х частей и часто встречающемуся на протяжении эволюции разработки программ. Как уже говорилось, этот шаблон включает перечисленные ниже моменты.

- Разработка каркасов для запуска реализаций продуктов на основе общих архитектурных стилей.
- Разработка инструментов на базе языка для поддержки разработки продуктов посредством адаптации, конфигурирования и сборки каркасных компонентов.
- Использование инструментов для вовлечения заказчиков и быстрой реакции на изменения требований за счет инкрементного построения программ с сохранением их работоспособности по мере внесения изменений.
- Фиксация проектных решений в форме, непосредственно порождающей исполняемые программы.

Мы приводили примеры приложения этого шаблона к двум доменам: конструирование интерфейса пользователя и дизайн базы данных. Затем мы отметили, что применение этого шаблона исключительно дорого, и из-за своей стоимости экономически оправдано в широких горизонтальных доменах, таких как конструирование пользовательского интерфейса и доступ к данным. И, наконец, мы предположили, что новые технологии могут использоваться для существенного снижения затрат, обеспечивая экономическую возможность получения повышенной производительности в узких вертикальных доменах вроде здравоохранения или финансовых услуг.

Разрабатывать инструменты на базе языка полезно, но дорого.

Две части этого шаблона уже экономически эффективны, а именно: разработка каркасов и использование гибких методов вовлечения заказчиков и быстрой реакции на изменения в требованиях. Вспомним из главы 3, однако, что давление требования быстрой поставки результатов может затруднить разработку каркасов, которые обычно должны быть результатом обобщения опыта, накопленного во многих проектах. Другие две части не так экономически эффективны. Согласно Робертсу (Roberts) и Джонсону (Johnson), применение хорошо спро-

ектированного шаблона может снизить стоимость разработки приложения в десятки раз [212]. Однако использование каркаса может оказаться трудным. Каркас представляет архитектурный продукт, такой как приложение или подсистема, который может быть дополнен или специализирован для удовлетворения разнообразных требований. Отображение требований каждого варианта продукта на каркас — нетривиальная проблема, обычно требующая экспертизы архитектора или ведущего разработчика. Робертс и Джонсон указывают, что инструменты на базе языка могут инкапсулировать абстракции, определенные каркасом, помогая пользователям думать в терминах этих абстракций.

Фабрики программного обеспечения используют инструменты на базе языка для отображения вариаций требований на дополнения каркаса.

Фабрики программного обеспечения используют языковые инструменты для отображения вариаций требований на дополнения каркаса. Это отображение — форма руководства, представленная архитекторами или ведущими разработчиками во время разработки фабрики ПО, и затем повторно используемая на протяжении разработки продукта — часто силами гораздо менее опытных разработчиков. Языковые инструменты также обеспечивают подвижность, фиксируя требования в формах, более понятных пользователю, и быстрее распространяя изменения требований в реализации. К сожалению, разработка таких инструментов для автоматизации процесса сборки в настоящее время выходит далеко за пределы доступного большинству организаций. Если эта часть шаблона сможет быть реализована столь же экономически эффективно, как и другие, то мы приблизимся вплотную к воплощению описанного представления.

Что такое фабрика программного обеспечения?

Мы определяем термин "фабрика программного обеспечения".

Подробный ответ на этот вопрос займет остаток этой книги. Однако мы можем представить краткое определение для обеспечения привязки к последующей дискуссии.

> ***Фабрика программного обеспечения*** *— это* ***линейка программных продуктов****, которая конфигурирует расширяемые инструменты, процессы и содержимое с использованием шаблона фабрики ПО, основанного на* ***схеме фабрики программного обеспечения****, для автоматизации разработки и поддержки вариантов первоначального продукта за счет адаптации, сборки и конфигурирования основанных на каркасе компонентов.*

История термина

Согласно Мириам Вебстер (Meriam Webster), фабрика — это высокоорганизованное средство производства, которое производит членов линейки продуктов, используя для этого стандартизованные части, инструменты и производственные процессы. Исторически термин "фабрика программного обеспечения" использовался для описания крупных коммерческих инициатив по автоматизации разработки программ одного направления. Важнейшими результатами этих усилий стали систематическое повторное использование и непрерывное совершенствования процесса. Согласно Ааину (Aaen), этот термин означает опыт долговременных усилий по интеграции и оптимизации методов и приемов разработки программного обеспечения, накопленный на протяжении разработки множества проектов [3].

Центральными элементами фабрики ПО являются схема фабрики ПО и шаблон фабрики ПО, основанный на этой схеме. Шаблон фабрики ПО конфигурирует расширяемые инструменты, процессы и содержимое, формируя возможности для производства семейства продуктов. Схематическое изображение фабрики ПО приведено на рис. 5.3.

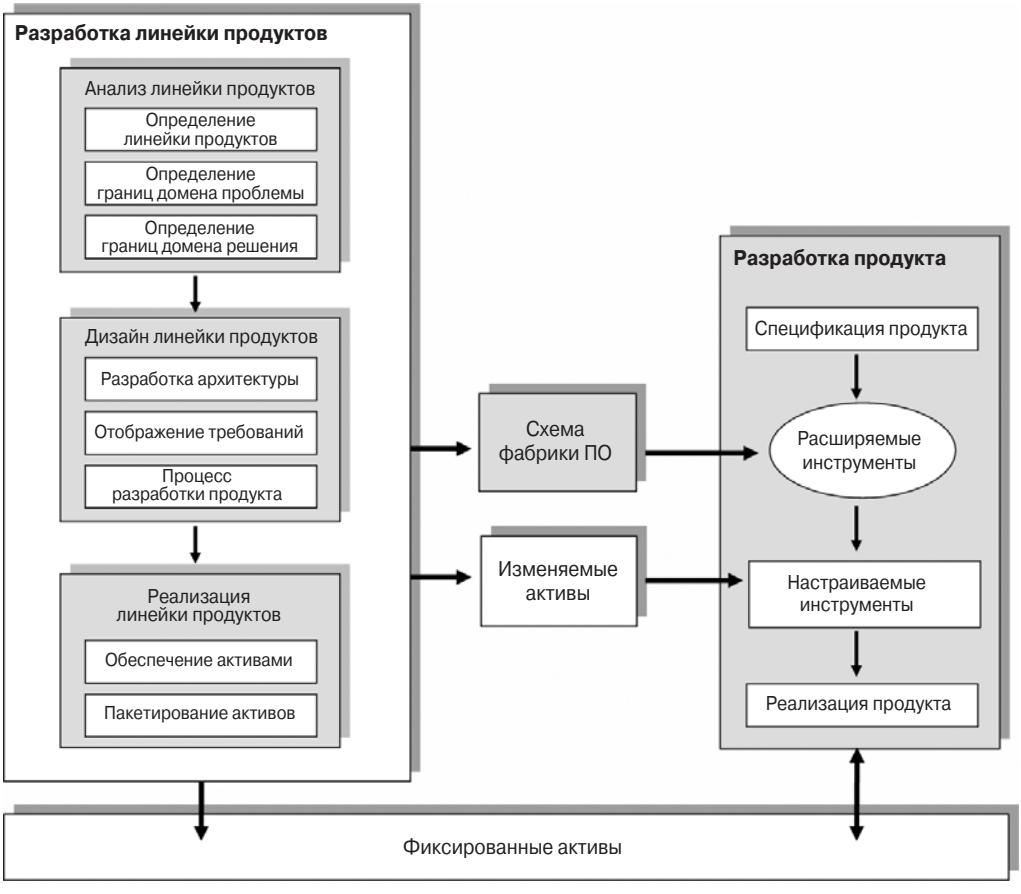

Рис. 5.3. *Фабрика программного обеспечения*

Мы используем эту диаграмму для рассмотрения каждой части фабрики программного обеспечения по очереди. Затем мы поговорим о том, как построить фабрику ПО, и как построить программный продукт, используя фабрику ПО.

Что такое схема фабрики программного обеспечения?

Используйте разные модели для описания разных точек зрения и разных уровней абстракции.

Необходимость в схеме фабрики программного обеспечения становится явной, когда мы осознаем потребность в способе категоризации и суммирования артефактов разработки, таких как XML-документы, модели, конфигурационные файлы, сценарии сборки, файлы исходного кода, SQL-файлы, файлы локализа-

ции, манифесты развертывания и описания сценариев тестирования, причем строго упорядоченным способом, а также потребность в способе определения отношений между ними. Общий подход заключается в применении сетки, показанной на рис. 5.4. Столбцы определяют ограничения, а строки — уровни абстракции. Мы выбрали метки для столбцов и строк, которые интуитивно понятны большинству разработчиков, но другие могут выбрать более трудоемкий способ их пометки[1]. Каждая ячейка определяет перспективу, или *точку зрения*, от которой мы можем строить некоторые аспекты программного обеспечения. Например, для трехзвенного приложения одна ячейка может определять логический вид звена представлений, а другая — концептуальный вид звена данных. Как только сетка сконструирована, мы можем наполнить ее артефактами разработки для конкретного программного продукта.

■ Сценарии и случаи использования ■ Бизнес-цели и намерения	■ Бизнес-сущности и отношения	■ Бизнес-процессы ■ Производство служб	■ Распределение служб ■ Качество стратегии служб
■ Модели рабочих потоков ■ Определение ролей	■ Схемы сообщений и спецификации документов	■ Взаимодействия служб ■ Определения служб ■ Объектные модели	■ Типы логических серверов ■ Отображение служб
■ Спецификации процесса	■ Схемы баз данных ■ Стратегия доступа к данным	■ Детальный дизайн ■ Дизайн, зависимый от технологии	■ Физические серверы ■ Установленное программное обеспечение ■ Организация сети

Рис. 5.4. *Сетка для категоризации артефактов разработки*

Конечно, сетка может быть использована для построения более одного программного продукта. Прежде чем она будет наполнена конкретными артефактами разработки, в ней определяется перечень материалов, необходимых для построения членов семейства программных продуктов.

Схема фабрики программного обеспечения определяет артефакты и активы, использованные для ее построения.

Обратившись к линейкам продуктов, мы можем сделать шаг вперед и добавить информацию к каждой ячейке, идентифицирующей производственные активы, которые будут применяться

[1] Сетка, описанная Джоном Зачманом (John Zachman) в его архитектурном каркасе предприятия (Enterprise Architecture Framework), содержит шесть столбцов и пять строк. См. *www.zifa.com*.

для построения артефактов разработки, требуемых в этой перспективе, включая такие, как DSL, шаблоны, каркасы и инструменты. Если мы также идентифицируем микропроцессы, используемые для каждой ячейки, то сможем воспринимать эту сетку как процессный каркас для производства продуктов, являющихся членами семейства.

Применение сетки к линейке продуктов и отображение производственных активов является новшеством.

В самой сетке нет ничего нового. Инновация состоит в ее применении к семейству продуктов, идентифицируя производственные активы для каждой ячейки, и определяя отображений между и внутри ячеек, которые должны использоваться для частичной или полной автоматизации трансформаций модели, генерации кода, применения шаблонов, конструирования тестового окружения, компоновки пользовательского интерфейса, дизайна схемы базы данных и многих других задач разработки. Как мы видели, следует использовать только первоклассные артефакты разработки, основанные на строгих языках, подобных XML, C# и SQL, чтобы обеспечить эту автоматизацию. Для моделей это означает применение DSL, а не языков моделирования общего назначения, предназначенных только для документирования. В некоторых случаях артефакты, описанные точками зрения, являются моделями, но часто это не так. Они могут быть любыми исходными артефактами, основанными на формальных языках, таких как высокоуровневые сценарии рабочего потока, файлы исходного кода на языках общего назначения, файлы WSDL или SQL-файлами языка определения данных (DDL) .

Точки зрения также накладывают требования на описываемые ими артефакты и могут использовать языки других точек зрения.

Обратите внимание, что точки зрения определяют не только языки, используемые для разработки описываемых ими артефактов, но также и требования к артефактам, обычно выражаемые в виде ограничений или шаблонов. Например, схема фабрики программного обеспечения может содержать две точки зрения, причем обе будут использовать один и тот же DSL моделирования классов. Мы можем потребовать, чтобы все классы, смоделированные в одной из этих точек зрения, наследовались прямо или непрямо от классов, принадлежащих к каркасу элементов управления пользовательского интерфейса, ассоциированного с этой точкой зрения. Аналогично мы можем потребовать, чтобы все классы, смоделированные с другой точки зрения, играли определенные роли в одном из нескольких шаблонов реализации бизнес-сущностей, ассоциированных с точкой зрения. В реальной схеме фабрики программного обеспечения одни и те же языки обычно используются многими точками зрения — особенно близкими к платформе и основанными на языках программирования общего назначения.

Схемы фабрик программного обеспечения как графы

Схема фабрики программного обеспечения — на самом деле ориентированный граф, узлы которого являются точками зрения, а стороны — вычисляемыми отношениями между ними.

Конечно, даже очень подробно детализированная сетка — всего лишь значительно упрощенное отображение реальности. Схема фабрики программного обеспечения — на самом деле ориентированный граф, узлы которого — это точки зрения, а стороны — вычисляемые отношения между ними, которые называются отображениями. Эта структура допускает существование узлов, которые не обязательно должны находиться рядом, чтобы быть связанными. Она также ослабляет искусственные ограничения, накладываемые представлением в виде сетки, ко-

торое предполагает четкую схему классификации в виде строк и столбцов. И, наконец, что наиболее важно — она позволяет схеме отражать программную архитектуру. Поэтому, например, схема для семейства бизнес-приложений может содержать несколько кластеров точек зрения — по одному для каждой подсистемы, такой как управление заказчиками, управления каталогами, управление заказами или их заполнение. Точки зрения в каждом кластере затем могут быть сгруппированы в подмножества, отражающие многослойную архитектуру этой подсистемы, как показано на рис. 5.5. Мы применяем эту схему фабрик программного обеспечения в качестве основы для нашего примера фабрики ПО в главе 16. Конечно, сетка — удобное упрощение, которое легко визуализировать. Поэтому мы будем использовать оба представления, подходящие для наших целей, и даже будем применять их взаимозаменяемым образом. Имейте в виду, однако, что граф — наиболее точное представление, а сетка — лишь удобная абстракция графа.

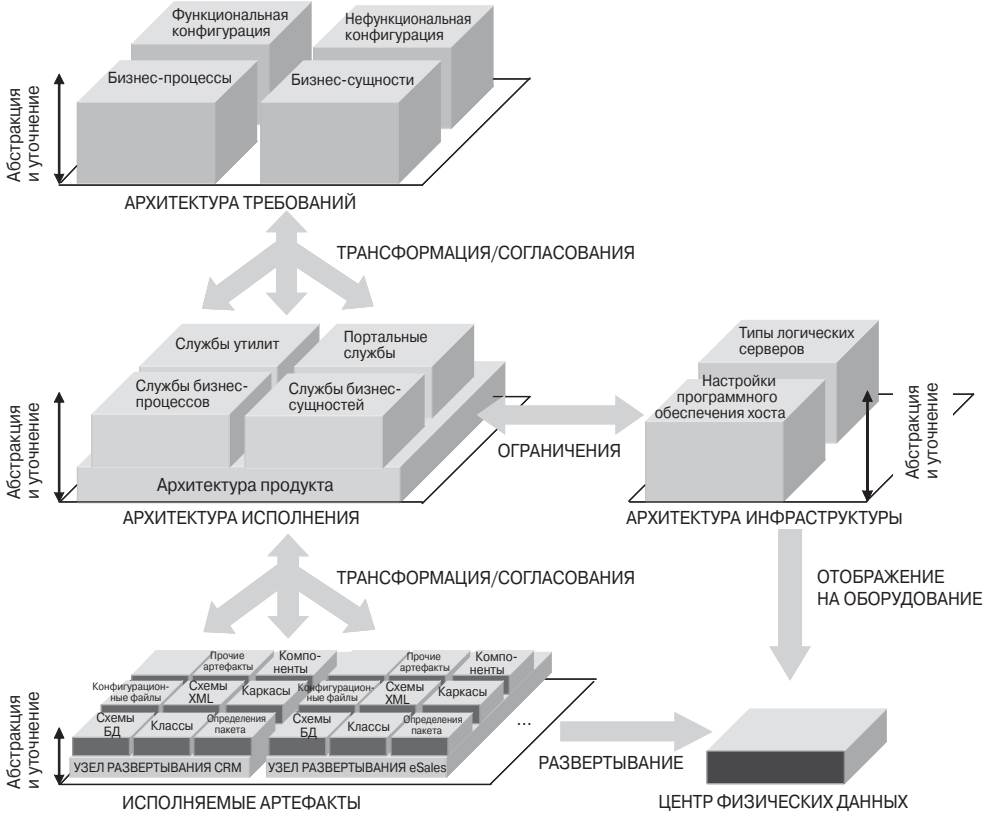

Рис. 5.5. *Схема фабрики программного обеспечения*

Мы называем граф *схемой фабрики программного обеспечения*, потому что она описывает артефакты, которые должны быть разработаны для производства программного продукта, подобно тому, как схема XML описывает элементы и атрибуты, которые могут быть созданы для формирования документа, и как схема

Схема фабрики программного обеспечения описывает артефакты, которые могут быть разработаны для производства программного продукта.

базы данных описывает строки, которые должны быть созданы для наполнения этой базы данных.

Теперь мы можем видеть, как схема фабрики программного обеспечения связывает вместе важнейшие новшества, описанные в главе 4. Подобно стандарту описания архитектур (Architectural Description Standard — ADS), схема фабрики программного обеспечения — это шаблон для описания членов семейства программных продуктов. Несмотря на это сходство, однако, есть существенные отличия между схемой фабрики программного обеспечения и ADS.

- В то время как ADS имеет дело только с архитектурой, схема фабрики программного обеспечения имеет дело со многими другими аспектами семейства программных продуктов, такими как требования, реализация, развертывание, тестирование, инструментальное обеспечение, отладка, управление, сопровождение и расширение.
- В то время как ADS организует проектную документацию, схема фабрики программного обеспечения организует артефакты разработки.
- В то время как ADS подразумевает семейство программных продуктов, он не идентифицирует его явно и не включает механизмов поддержки разработки на базе семейств, таких как способ выражения того, чем члены семейства отличаются от оригинала этого семейства. Схема фабрики программного обеспечения, с другой стороны, нацелена на специфическое семейство программных продуктов, и может служить основой для создания экземпляров и настройки на описание конкретного члена семейства в терминах отличий от оригинала семейства.
- В то время как ADS не обязательно поддерживают автоматизацию, схема фабрики программного обеспечения может быть реализована с помощью шаблона фабрики ПО для автоматизации задач разработки программного обеспечения, в чем мы вскоре убедимся.

Схема фабрики программного обеспечения представляет многомерное разделение отношений.

Конечно, существенное свойство схемы фабрики программного обеспечения заключается в том, что она представляет многомерное разделение отношений, основанных на разных аспектах организуемых артефактов, таких как уровень абстракции, положение внутри архитектуры, функциональные или эксплуатационные качества. Согласно Коплиену (Coplien) [55]:

> *Можно анализировать домен приложения, используя принципы обобщения и вариации для разделения его на поддомены, каждый из которых может подойти для дизайна, соответствующего определенной парадигме.*

Схема фабрики программного обеспечения подобны рецептам, определяющим ингредиенты, инструменты и подготовительные процессы для семейства программных продуктов.

Теперь мы можем воспринимать сетку как двумерную проекцию графа, отображающего один или более аспектов по горизонтальной оси и разные уровни абстракции — на вертикальную ось. Другая двумерная проекция — плоскость аспектов, проектирующая взаимосвязанные точки зрения графа на специфический аспект, и обеспечивая консолидированный взгляд с этой точки зрения. Примерами плоскостей аспектов являются плоскости дистрибуции, безопасности и транзакций.

Мы также воспринимаем схему фабрики программного обеспечения как рецепт для построения семейства программных продуктов. Ясно, что точка зрения определяет ингредиенты и инструменты, применяемые для их подготовки, но где описыва-

ется сам процесс подготовки? Вспомните из главы 4, что процессный каркас может быть сконструирован определением микропроцессов, необходимых для разработки каждого члена в наборе взаимосвязанных артефактов и определением ограничений, касающихся предварительных условий, которые должны быть удовлетворены перед производством артефакта, постусловий, которые должны быть удовлетворены после его производства, а также инвариантов, которых должен придерживаться артефакт после стабилизации. Этот каркас определяет пространство возможных процессов, которые могут возникать, в зависимости от потребностей и обстоятельств данного проекта. Определенно, есть некоторое сходство между процессным каркасом и схемой фабрики программного обеспечения. В самом деле, точки зрения схемы фабрики ПО уже определяют микропроцессы для производства описанных ими артефактов. Это отношение означает, что мы можем получить процессный каркас из схемы фабрики ПО, просто добавляя ограничения, регламентирующие порядок выполнения. Теперь мы имеем полный рецепт для членов семейства продуктов. Он определяет ингредиенты, инструменты, используемые для их приготовления, и сами процессы приготовления.

Отношения между точками зрения

При изменении артефактов необходимо поддерживать согласованность.

Обычно, когда мы описываем продукт, используя множественные артефакты, информация, зафиксированная одним артефактом, может быть связана с информацией, зафиксированной другим. Например, модель, описывающая схему базы данных, может быть уточнением модели, описывающей бизнес-сущности. Аналогично информация, зафиксированная одним артефактом, может перекрываться информацией, зафиксированной другими. Например, модель структуры бизнес-сущности может ссылаться на те же классы, что и модель, описывающая политику безопасности для всех бизнес-сущностей в сборке. Ясно, что необходимо определить отношения между артефактами для поддержки согласованности при проведении изменений. Например, если в предыдущем примере имя бизнес-сущности изменяется в одной или двух моделях, оно должно измениться и в остальных.

Отношения между артефактами определяются отношениями между точками зрения.

Мы определяем отношения между артефактами, определяя отношения между точками зрения, описывающими эти артефакты. Отношения между точками зрения определены в терминах отображений. Отображение инкапсулирует знания о том, как реализовать артефакты, описанные одной точкой зрения, в терминах артефактов, описанных другой, и делает эти знания доступными для повторного использования. Опережающее отображение определяет, как артефакты, описанные целевой точкой зрения, наследуются от артефактов, описанных исходной точкой зрения. Обратное отображение определяет подобное отношение, но в обратном направлении. В каждом направлении наследование может быть как частичным, так и полным, и должно дополняться, используя информацию из некоторого другого источника.

Вычисляемые отображения могут применяться для генерации артефактов.

Если отображение вычисляемое, то мы можем частично или полностью генерировать артефакты, описанные целевой точкой зрения, из артефактов, описанных исходной точкой зрения. Чтобы определить вычисляемое отображение, необходимо иметь формальное описание как исходной, так и целевой точки зрения, а это требует применения формальных языков для выражения описываемых артефактов. Язык исходной точки зрения определяет элементы,

которые могут появляться в описанных артефактах, и виды выражений, которые могут формироваться их комбинациями. Язык целевой точки зрения определяет элементы, которые могут использоваться для реализации выражений в артефактах, описанных исходной точкой зрения и правила их комбинации.

Вычисления между отображениями называются трансформациями.

Вычисления между отображениями называются трансформациями. Трансформации читают выражения от одного или более исходных артефактов. Трансформации могут быть полностью ручными, полностью автоматическими или промежуточными, в зависимости от объема требуемой изменчивости. Если требуется бесконечная изменчивость, то трансформация должна быть полностью ручной. Если никакой изменчивости не нужно, трансформация может быть полностью автоматической. Частично автоматизированные трансформации могут включать интерактивное конструирование результатов, субъектов для ограничений, продиктованных исходными артефактами. Например, мы можем пожелать вручную спроектировать пользовательский интерфейс, ограниченный требованием, чтобы каждое поле формы отображалось на свойство объекта, возвращенного бизнес-операцией, и что каждое свойство должно визуализироваться полем. В прочих случаях мы можем проверить результаты трансформации и изменить исходные артефакты или параметры трансформации, чтобы изменить результаты. Некоторые изменения могут быть выполнены непосредственно над результатами для оптимизации или настройки. Некоторые из них могут быть локальными по отношению к результатам, в то время как другие — отображаться обратно на исходные артефакты. Мы вернемся к изучению отображений и трансформаций, основанных на них, в главах 14 и 15.

Настройка отношений

Возможность настройки отображений важна для высокоточной трансформации и генерации кода.

Возможность настраивать отношения очень важна для точной трансформации и генерации кода. Например, существуют три широко известных способа представить наследование при сохранении объекта в базу данных, как показано на рис. 5.6. (На этом рисунке PK означает “первичный ключ”, а PFK — “родительский внешний ключ”.) Верхний сохраняет экземпляры каждого класса в отдельной таблице. Средний сохраняет только дополнительные столбцы, используемые классом `Class2`, в отдельной таблице. Нижний сохраняет экземпляры обоих классов в единственной таблице, используя дискриминатор `T_Class12_1D` для идентификации класса объекта, сохраняемого в каждой строке, и помещая NULL в столбцы для неиспользуемых атрибутов в строках, содержащих экземпляры класса `Class1`. Каждое из этих представлений обеспечивает оптимальную производительность для определенных шаблонов доступа. Чтобы настроить производительность системы постоянства объектов, нужно иметь возможность настраивать отображения между моделями бизнес-сущностей и схемами базы данных. Эта диаграмма основана на нотации, предложенной Найлбургом (Nailburg) и Максимчуком (Maksimchuk) [171].

Для настройки трансформаций доступно несколько механизмов.

Установка значений параметров — простейшая форма настройки, как показано на рис. 5.7, на котором представлен экранный снимок IBM Rational Rose XDE. Параметры удобны, потому что они четко фиксируют проектные решения. Шаблоны предоставляют возможность более развитой настройки с минимальными усилиями. Трансформации на базе шаблонов описа-

ны в главах 7 и 15. Еще более развитая настройка может быть обеспечена применением расширений компилятора, таких как шаблоны, сценарии и компилированные расширения. Некоторые из этих механизмов рассматриваются в главе 7.

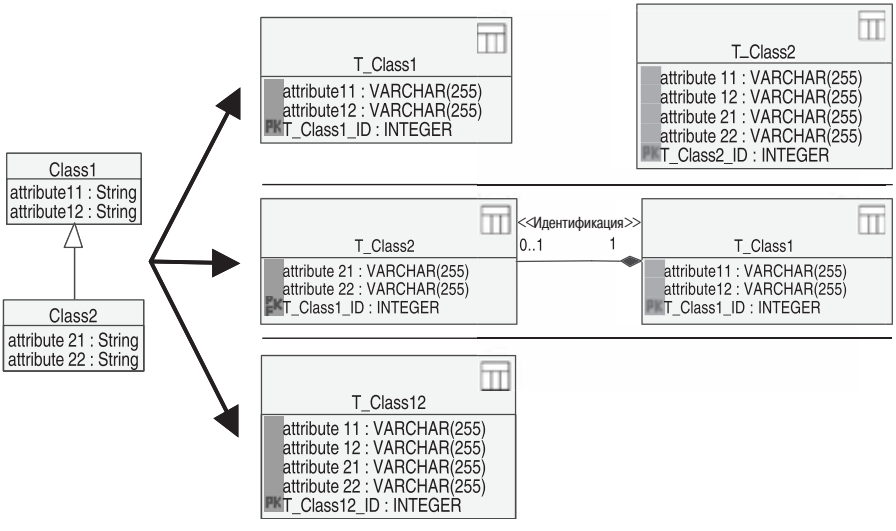

Рис. 5.6. *Три способа представления наследования*

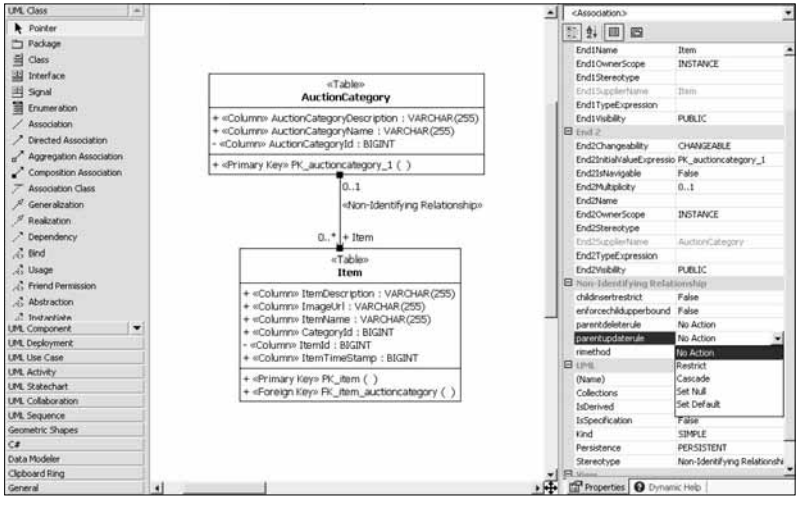

Рис. 5.7. *Настройка отношений*

Синхронизация артефактов

Синхронизация с применением отображений предназначена для сохранения согласованности артефактов.

Как только у нас появляются два или более взаимосвязанных артефакта, мы можем использовать трансформации для обеспечения их согласованности во время разработки. Изменения в артефактах или значениях параметров могут сделать отношения между ними недействительными. Когда такое случается, трансформация должна быть повторно вычислена, либо несогласованность должна быть помечена, дабы пользователь откорректировал ее вручную. Повторное автоматическое вычисление трансформации во время работы пользователя называется синхронизацией. Синхронизация позволяет избежать ручной работы по обеспечению согласованности артефактов. Например, модель бизнес-сущностей и схема базы данных должны быть синхронизированы так, чтобы когда изменения проводятся в первой, они автоматически отображались на вторую.

Даже частичная синхронизация является усовершенствованием по сравнению с ручным управлением.

Модель, которая работает как источник синхронизированного обновления в одном случае, может служить целью в другом. Синхронизация может происходить немедленно после изменения участвующего артефакта, либо она может осуществляться в определенные моменты времени, такие как момент изменения фокуса в инструменте или только по явному запросу. Когда артефакты синхронизированы, может стать невозможно разрешить разницу между ними автоматически. В этом случае мы можем сравнить два артефакта и объединить изменения. Даже частичная синхронизация является усовершенствованием по сравнению с ручным управлением согласованностью, когда элементы, затронутые изменением, должны быть идентифицированы посредством визуального просмотра, а также ошибки компиляции и дефекты, проявляющиеся во время выполнения. Другие цели синхронизации включают ограничение пользовательских действий лишь теми, которые сохраняют корректные отношения между артефактами и синхронизируют их подмножества вроде сигнатур методов, но не их тел.

Конфигурирование схемы фабрики программного обеспечения

Мы должны сконфигурировать схему фабрики программного обеспечения для построения любого данного члена семейства продуктов.

Важный аспект использования схемы фабрики ПО — это конфигурация. “Сырая” схемы фабрики ПО — просто рецепт для построения семейства программных продуктов. Однако в любого данном проекте мы заботимся только об одном программном продукте — том, построением которого занимаемся в данный момент. Поэтому мы должны сконфигурировать схему фабрики ПО для создания рецепта построения определенного члена семейства продуктов. Этот процесс подобен тому, что делает повар, когда модифицирует рецепт перед его использованием на основе специальных требований, таких как количество персон, доступные ингредиенты и инструменты, и время, отпущенное на приготовление блюда. Эти модификации обычно достаточно очевидны и состоят большей частью из конфигураций, как будет показано в главах 11 и 16. Мы можем теперь видеть, что для того, чтобы поддерживать конфигурацию такого рода, схема фабрики ПО должна содержать как фиксированные, так и изменяемые части. Фиксированные части остаются одинаковыми для всех членов семейства продуктов, в то время как остальные — меняются в соответствии с уникальны-

ми требованиями. Например, схема фабрики ПО для системы онлайновой регистрации заказов (Online Order Entry Systems) должна включать точки зрения, описывающие артефакты и инструменты, применяемые для построения персонализованных подсистем. Однако для некоторых членов семейства персонализованная подсистема может и не понадобиться. Прежде чем мы соберемся строить такую систему, сначала нужно удалить точки зрения, связанные с конструированием персонализованных подсистем, конфигурируя структуру проекта и инструменты соответствующим образом.

Что такое шаблон фабрики программного обеспечения?

Необходим шаблон фабрики программного обеспечения, который реализует схему фабрики ПО для построения члена семейства.

Итак, наш процесс фабрики программного обеспечения описан достаточно привлекательно, но пока еще только на бумаге. Если все, что у нас есть — это схема фабрики ПО, то мы можем описать активы, используемые для построения членов семейства, хотя самих активов не имеем. Чтобы действительно построить член семейства продуктов, необходимо реализовать схему фабрики ПО, определив DSL, шаблоны, каркасы и инструменты, которые в ней описаны, упаковав их и обеспечив доступ к ним для разработчиков продукта. Все вместе эти активы формируют *шаблон фабрики программного обеспечения*. Этот шаблон включает код и метаданные, которые могут быть загружены в расширяемые инструменты, подобные интегрированным средам разработки (IDE) или инструментальному комплекту жизненного цикла предприятия, с тем, чтобы автоматизировать разработку и поддержку членов семейства. Мы называем код и метаданные шаблонов фабрики ПО, потому что он конфигурирует инструменты для производства специфического типа программного обеспечения — точно так же, как шаблон документа, загруженный в инструмент персонального пользования типа Microsoft Word или Excel, конфигурирует его для производства документа специфического типа. Как и схема фабрики ПО, так и шаблон фабрики ПО должны быть сконфигурированы для построения специфического члена семейства. В то время как конфигурирование схемы настраивает описание фабрики ПО, конфигурирование шаблона настраивает инструмент и другие части среды разработки, используемой для построения члена семейства.

Построение фабрики программного обеспечения

Построение фабрики программного обеспечения — это специальный случай построения линейки программных продуктов.

Теперь мы готовы к тому, чтобы поговорить о построении фабрики ПО. Поскольку фабрика ПО — это разновидность линейки программных продуктов, построение ее является специальным случаем построения линейки программных продуктов. Более углубленная дискуссия о разработке линеек программных продуктов подождет до главы 11. Мы будем использовать определенные там термины и приведем краткое объяснение некоторых из них. Построение фабрики ПО включает в себя перечисленные ниже действия.

- Построение схемы фабрики ПО, описывающей фабрику. Это специализация двух действий, связанных с линейками продуктов:

- *Анализ линейки продуктов* определяет, какие продукты будет выпускать фабрика. Основное содержание этого действия — определение области, идентифицирующей линейку программных продуктов, которые будут разработаны и поддержаны с использованием фабрики программного обеспечения. Эта область определяется не перечислением описаний специфических продуктов, а скорее описанием проблемного домена, на который ориентированы эти продукты. Анализ линейки продуктов порождает, помимо прочих вещей, требования к линейке продуктов. Требования к линейке продуктов организованы по точкам зрения, которые становятся частью схемы фабрики продуктов.
- *Дизайн линейки продуктов* определяет, как фабрика ПО будет разрабатывать продукты в пределах своей области. Главное содержание этого действия — определение архитектуры для целевого семейства программных продуктов. Эта архитектура подобна архитектуре отдельного продукта, за исключением того, что она предназначена для поддержки вариаций между членами семейства. Дизайн линейки продуктов производит несколько других артефактов, включая требования отображения, о которых мы поговорим ниже, процесс разработки программного обеспечения и план использования инструментов для автоматизации частей процесса разработки продукта. Эти артефакты также вносят вклад в схему фабрики ПО: архитектура организована по точкам зрения, требования отображений выражаются с использованием отношений между точками зрения, а процесс разработки продукта выражен как набор микропроцессов, прикрепленных к точкам зрения.

➠ Построение шаблона фабрики ПО, который создает экземпляр линейки программных продуктов. Это специализация реализации линейки продуктов. Шаблон фабрики продуктов содержит в себе производственные активы для фабрики ПО, включая активы требований, такие как спецификация языков, активы реализации, такие как шаблоны, каркасы и компоненты, активы процессов, такие как средства разработки, активы тестирования, такие как тестовые окружения, тесты узлов и интегрированные тестовые комплекты и активы развертывания, такие как конфигурации логических хостов, на которых члены семейства будут развертываться. Шаблон фабрики программного обеспечения должен обеспечивать отсутствие значительных расхождений между артефактами, описанными с разных точек зрения, обеспечивать, что любые видимые артефакты, использованные фабрикой ПО, могут изменяться, и что не будет необратимых шагов в процессе разработки продукта.

Мы сопровождаем ее по мере разработки членов семейства.

Как только заложен первый угол фабрики ПО, затем его следует поддерживать постоянно, накапливая отзывы по мере разработки членов семейства, которые мы можем применять для уточнения его определения и реализации. Как мы говорили ранее, наиболее дорогостоящая часть разработки фабрики ПО — это построение языков и инструментов, призванных автоматизировать разработку продуктов. Конечно, есть много других существенных сложностей вроде выполнения необходимого анализа доменов, необходимого для выбора соответствующей схемы фабрики ПО, а также разработки инфраструктуры, требуемой для поддержки создания, конфигурирования и инсталляции шаблонов фабрики ПО. Анализ домена мы обсудим в главе 11, разработку инфраструктуры — в главах 14 и 15, а технологии, облегчающие, ускоряющие и удешевляющие разработку инструментов — в главах 8 и 9.

Построение программного продукта

Построение продукта с применением фабрики ПО — это специальный случай построения продукта с использованием линейки продуктов.

Конечно, целью построения фабрики ПО является быстрая разработка членов семейства продуктов. Как вы, вероятно, предполагаете, построение продукта с применением фабрики ПО — это специальный случай построения его с использованием линейки программных продуктов. Углубленная дискуссия на эту тему опять же может подождать до главы 11, но мы опять же начнем использовать термины, определенные там, и приведем здесь краткое пояснение некоторых из них. Построение продукта с использованием фабрики ПО включает перечисленные ниже действия.

- *Анализ проблемы* определяет, находится ли продукт в области применения фабрики ПО. В зависимости от ответа, мы можем отдать предпочтение построению его части или всего продукта целиком за пределами фабрики ПО. Мы можем также предпочесть построение не очень подходящего продукта в фабрике ПО, изменив для этого ее схему и шаблон.
- *Спецификация продукта* определяет требования продукта в терминах отличий от требований линейки продуктов. Может быть использован целый диапазон механизмов спецификации продуктов, в зависимости от ширины отличий в требованиях, включая панели свойств, интерактивные мастера, модели средств, визуальные модели или прозаические табличные и иерархические структуры.
- *Дизайн продукта* отображает отличия в требованиях на отличия в архитектуре линейки продуктов и процесс разработки продукта, порождая настраиваемую архитектуру продукта и настраиваемый процесс его разработки.
- *Реализация продукта* включает знакомые действия, такие как разработку и построение компонентов и узловых тестов, выполнение этих тестов и сборку компонентов. Может быть использован целый диапазон механизмов спецификации продуктов, в зависимости от ширины отличий в требованиях, включая панели свойств, интерактивные мастера, модели средств, конфигурирующих компоненты, визуальные модели, собирающие компоненты и генерирующие другие артефакты, включая модели, код и файлы конфигурации, а также исходный код, дополняющий каркасы, либо создающий, модифицирующий, расширяющий или адаптирующий компоненты.
- *Развертывание продуктов* включает создание или повторное использование ограничений развертывания по умолчанию, конфигурацию логического хоста и отображения исполняемой программы на логический хост, реконфигурацию хостов посредством установки и конфигурирования необходимых ресурсов, а также инсталляции и конфигурирования развертываемых исполняемых программ.
- *Тестирование продуктов* включает создание или повторное использование тестовых активов, включая сценарии тестирования, окружения, наборы тестовых данных, а также применение инструментальных и измерительных инструментов.

Конечно, большинство из этих шагов, если не все, должны быть выполнены при разработке программного продукта — независимо от того, используется фабрика ПО или нет. Применение фабрики ПО гарантирует выполнение работы в определенном порядке,

под управлением процессного каркаса. Оно также уменьшает объем работы, которую необходимо выполнить, за счет повторного использования существующих производственных активов.

Схема фабрики ПО и ее шаблон могут быть сконфигурированы или реконфигурированы в любой момент.

Производственные активы, включая требования, процесс разработки, архитектуру, компоненты, конфигурацию развертывания и тесты, специфицируются, повторно используются, управляются и организуются от многих точек зрения, как определено схемой фабрики ПО. Схема фабрики ПО может быть сконфигурирована или реконфигурирована в любой момент за счет добавления, уничтожения или модификации точек зрения и отношений между ними. Шаблон фабрики ПО также должен конфигурироваться наряду с ее схемой. Примеры конфигураций шаблонов фабрик ПО включают выбор подсистем и компонентов для разработки, выбор шаблонов для применения и каркасов для расширения, модификацию структуры и политик проекта, а также конфигурирование исполняющей среды. Например, разрешение персонализации содержимого для приложения онлайновой коммерции может привести к перечисленным ниже моментам.

- Папка для подсистемы персонализации добавляется к проекту построения приложения.
- Каркас и сопровождающие его шаблоны для подсистемы персонализации импортируются в проект.
- Точка зрения, используемая для конфигурирования персонализации, добавляется к схеме, заставляя инструмент конфигурирования персонализации появиться в меню.
- Шаблон "Контроллер переднего плана" (Front Controller) применяется автоматически в трансформации между моделью взаимодействия с пользователем и моделью дизайна переднего плана Web, и появляется в дизайнере, в котором моделируется этот самый дизайн переднего плана Web. Причем он применяется вместо шаблона "Контроллер страницы" (Page Controller), так что запуск страницы переадресует разных пользователей на разные страницы, вместо того, чтобы показывать одно и то же содержимое для всех пользователей.
- Политика проекта в папке слоя презентации модифицируется, так что невозможно создать класс, унаследованный от PageController.

Процесс разработки

Мы можем работать в любом направлении по своему выбору, при условии, что удовлетворены ограничения процесса, определенные в схеме фабрики ПО.

Разработка продукта организуется и ограничивается процессом разработки, определенным при создании самой фабрики ПО, и затем подгоняется под разрабатываемый продукт во время конфигурирования схемы фабрики ПО. Можно работать сверху вниз от требований к исполняемой программе, сохраняя связанные артефакты синхронизированными. Например, мы можем разработать модель бизнес-сущности, воспользоваться ею для производства логической модели данных, а затем на ее основе произвести оптимизированную схему базы данных. Мы можем также воспользоваться ограничениями, определенными в горизонтальных отображениях между точками зрения.

Например, информация о среде развертывания (то есть доступные протоколы и службы) может быть использована для ограничения дизайна взаимодействий служб (ограничивая их протоколами и службами, доступными в среде развертывания). Можно также работать снизу вверх, создавая сначала тестовые окружения и специализированные компоненты для различных частей продукта и тестируя их в процессе работы. Когда схема фабрики программного обеспечения наполнена — процесс завершен. Можно работать любым способом по своему усмотрению, в соответствии с потребностями проекта, окружением и возникающими обстоятельствами, но при условии соблюдения ограничений процесса, определенных в схеме фабрики ПО. Обратите внимание, что разработка программ снизу вверх происходит в основном во время реализации фабрики ПО — по мере того, как создаются языки, шаблоны, каркасы и компоненты для поддержки абстракций разработки продуктов. Однако определенный объем работы снизу вверх всегда возникает при разработке продуктов, поскольку каждый продукт имеет уникальный набор требований, которые могут не быть предусмотрены заранее разработанными абстракциями. Как только сконфигурированная схема фабрики ПО адекватно наполнена, а это означает, что разработаны артефакты для рабочих подмножеств точек зрения, появляется возможность быстрой модификации реализации за счет изменения этих артефактов. Быстрый отклик позволяет масштабировать систему до значительного уровня сложности, не теряя при этом подвижности. В дополнение к быстрой модификации, другим преимуществом этого подхода является возможность применения инструментов для проверки корректности программного обеспечения по мере его эволюции, а также отслеживание и управление зависимостями между его компонентами.

Механизмы изменчивости

Мы говорили ранее, что широкий диапазон механизмов может быть использован для спецификации и реализации продуктов, в зависимости от размеров отличий между продуктами, находящимися в разработке и оригиналом продукта, определенным фабрикой ПО. Именно эта гибкость делает фабрики ПО настолько более производительными, чем одноразовая разработка. С фабриками ПО мы постепенно смещаемся от полностью открытого ручного кодирования, которое составляет основу разработки ПО в наши дни, к более ограниченным формам спецификации, полагающимся на предопределенные абстракции.

Смещение от полностью открытого ручного кодирования к более ограниченным формам спецификации — ключ к ускорению разработки программ.

Чернецки (Czarnecki) и Эйзенекер (Eisenecker) [58] определяют спектр механизмов изменчивости, простирающийся от рутинной конфигурации, через поэтапную конфигурацию на основе моделей средств, к творческому конструированию с использованием визуального моделирования, как показано на рис. 5.8.

Открытое ручное кодирование, не показанное на диаграмме, лежит справа от творческого конструирования с применением графического языка. Шаблоны представляют интересную точку в этом спектре между визуальным моделированием и открытым ручным кодированием, как описано в главе 6.

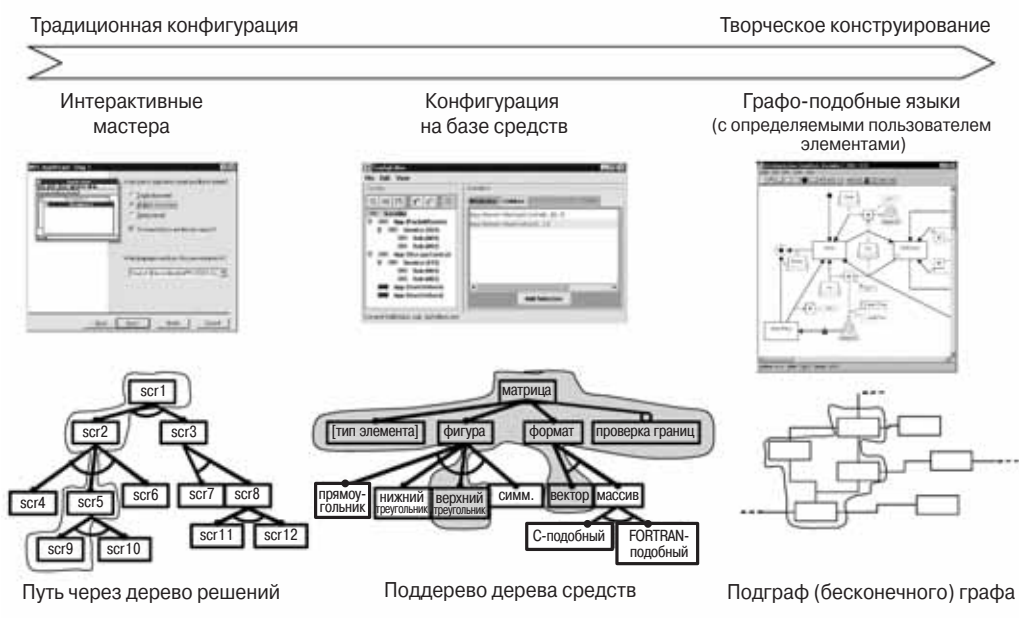

Рис. 5.8. *Спектр ограниченных механизмов изменчивости. Используется с разрешения Чернецки и Эйзенекера*

Пример фабрики программного обеспечения

Здесь мы приводим краткий пример. В главе 16 предлагается более детальный пример.

Здесь мы приведем лишь краткий пример, поскольку более детализированный пример будет приведен в главе 16. Предположим, что имеется независимый поставщик программ (Independent Software Vendor — ISV), который продает своим заказчикам приложения онлайновой регистрации заказов (Online Order Entry — OOE). Ответственные лица в компании работают с разработчиками линейки продуктов для определения границ семейства продуктов и оценки бизнес-прецедента. Убедившись в том, что цели достижимы, они решают инвестировать в фабрику ПО для снижения расходов и сокращения времени выхода на рынок с одновременным повышением качества. Разработчики линейки продуктов используют результаты анализа для быстрого запуска разработки схемы фабрики ПО, показанной на рис. 5.9. (Это правый нижний угол сетки из рис. 5.4, заполненный для фабрики ПО, используемой в данном примере.) На диаграмме прямоугольники представляют точки зрения, пунктирные линии представляют уточнения, а сплошные линии — ограничения.

Точки зрения описывают артефакты и активы, используемые для их построения.

Вспомните, что каждая точка зрения описывает нечто большее, нежели артефакты разработки. Она также описывает перечисленные ниже вещи.

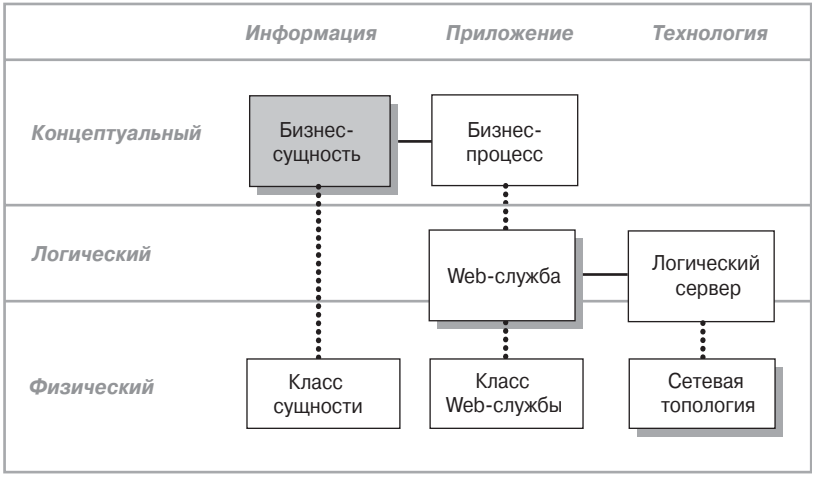

Рис. 5.9. *Простая схема фабрики программного обеспечения*

- DSL, применяемые для построения артефактов на основе точки зрения.
- Реструктуризации, которые могут усовершенствовать артефакты, основанные на точке зрения.
- Аспекты, применимые к артефактам, основанным на точке зрения.
- Микропроцессы, используемые для производства артефактов, основанных на точках зрения.
- Ограничения, накладываемые артефактами, основанными на связанных точках зрения.
- Каркасы, поддерживающие реализации артефактов, основанных на точках зрения.
- Отображения, поддерживающие трансформации внутри артефактов, основанных на точках зрения.

ISV определяет простую схему фабрики ПО.

Большинство точек зрения, показанных здесь, описывают модели, основанные на одном DSL. Единственными исключениями являются точки зрения класса сущности и класса Web-службы — они обе описывают файлы исходного кода C#. Рисунок иллюстрирует следующие моменты.

- Точка зрения “Бизнес-сущность” определяет абстракцию бизнес-сущности, описывающую эффективные, управляемые событиями, слабо связанные службы данных, которые отображаются на объектно-реляционный каркас. Примерами бизнес-сущностей могут служить “Заказчик” и “Заказ”.
- Точка зрения “Бизнес-процесс” определяет бизнес-действие, роль и абстракции зависимости, а также систематику шаблонов процессов, которые могут применяться для их составления, формируя спецификации бизнес-процесса. Примером бизнес-процесса может служить “Ввод согласованного заказа”, который может использовать три процессных шаблона: один для процесса пользовательского интерфейса — для наполнения покупательской тележки, один для последовательного процесса

подтверждения заказа и выполнения проверок кредитоспособности, а другой для управляемого правилами процесса вычисления скидки.

- Эти две точки зрения отображаются на точку зрения Web-службы, которая описывает кооперации Web-службы в архитектуре приложений, ориентированной на службы. Точка зрения Web-службы используется для описания того, как бизнес-сущности и процессы реализованы в виде Web-служб, как определены передаваемые ими сообщения и какие протоколы применяются для поддержки их взаимодействия, используемые абстракции, скрывающие детали реализации Web-службы. Шаблоны взаимодействия Web-службы, такие как "*фасад службы*", "*интерфейс службы*" и "*ворота*", могут быть применены для гарантии того, что архитектура приложения следует передовым приемам [169]. Политика безопасности может быть специфицирована как аспект и применена к каждой операции, определенной внутри любой группы выбранных служебных портов.
- Точка зрения "Архитектура логических систем" описывает конфигурации информационного центра данных. Она позволяет архитектору сети создавать масштабируемые инвариантные конфигурации центра данных в терминах логических серверов и соединений, инсталлированного программного обеспечения и конфигурационных установок. Все это станет целями для развертывания Web-служб. Стандартная конфигурация, подобная системной архитектуре Microsoft для информационного центра предприятия, показанной на рис. 5.10, может быть применена к моделям, основанным на этой точке зрения.

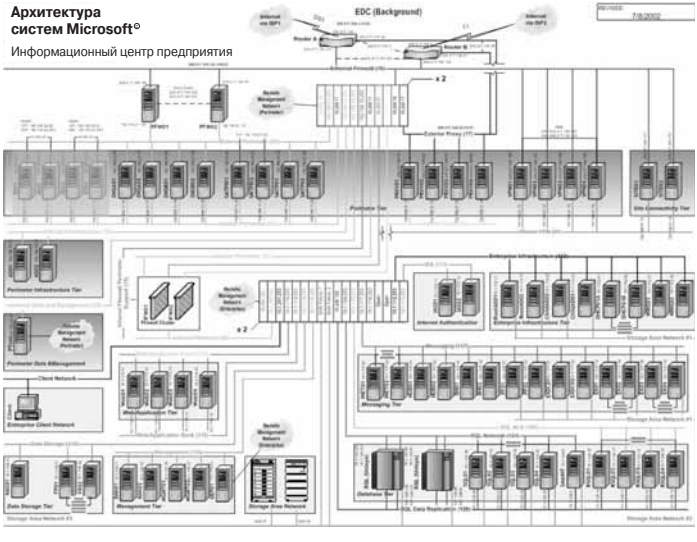

Рис. 5.10. *Логическая конфигурация информационного центра*

- Информация из одной модели влияет на разработку прочих. Примерами могут служить взаимодействия между бизнес-сущностями и процессами, а также между Web-службами и логическими серверами. Последнее особенно интересно, поскольку может быть использовано для дизайна развертывания. Встраивание знания об инфраструктуре развертывания в дизайн Web-службы ограничивает этот дизайн для предотвращения проблем с развертыванием. Аналогично, работая с этим в об-

ратном направлении, если Web-служба должна быть развернута на заданном логическом типе сервера, то мы можем проверить, что сервер, на котором она будет развернута, относится к правильному типу, что он имеет нужное инсталлированное программное обеспечение и правильно сконфигурирован. Этот подход, называемый дизайном для развертывания, проиллюстрирован на рис. 5.11, где показан снимок экрана Microsoft Visual Studio.

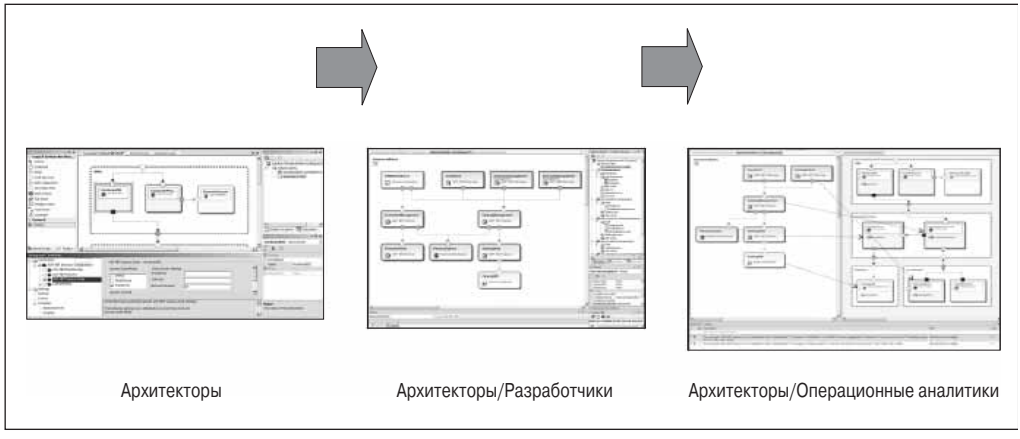

Рис. 5.11. *Дизайн для развертывания*

- Отображения между точками зрения поддерживают синхронизацию между моделями во время проектирования. Частичные реализации Web-служб генерируются из моделей в форме классов, которые дополняют каркас, таких как ASP.NET и разнообразных конфигурационных файлов и файлов политик.

По завершении схемы фабрики ПО разработчики линейки продуктов строят шаблон фабрики ПО на основе этой схемы для комплекта жизненного цикла приложения, используемого в компании для создания приложений. В качестве быстрого старта они используют ASP.NET и инструмент сборки Web-службы, разработанный ими для предыдущей фабрики ПО. Они экспериментируют с тестами и активами в процессе их разработки, применяя их для построения прототипа приложения. Накопление опыта работы с активами ведет к двум изменениям в схеме фабрики ПО — добавлению шаблонов к точке зрения “Класс Web-службы” и добавлению каркаса сущностей к точке зрения “Класс сущности”.

ISV строит шаблон фабрики ПО на основе схемы, строит некоторое прототипное приложение и затем выполняет ревизию схемы.

Когда компания готова к опробованию фабрики ПО, команда пилотного проекта загружает шаблон фабрики ПО в инсталляции комплекта жизненного цикла предприятия, конфигурируя его для быстрой разработки приложения онлайновой регистрации заказов. Далее команда пилотного проекта работает с ранее доступным партнером, чтобы определить приложение, которое необходимо построить. Они начинают с согласования временных рамок, затрат и самого процесса проекта. Когда

ISV работает с ранее доступным партнером для определения пилотного приложения и осуществления ревизии фабрики.

все это определено, фиксируются высокоуровневые требования, используя для этого инструменты специфицирования продукта, поставленные командой разработки линейки продуктов. Около 75% приложения может быть быстро разработано с использованием фабрики ПО, причем большая часть — посредством генерации кода. Остаток будет разработан вручную командой пилотного проекта. Команда пилотного проекта осуществляет ревизию схемы и шаблона фабрики ПО для подгонки той части приложения, которая должна разрабатываться вручную.

Команда пилотного проекта подгоняет требования и процесс разработки продукта, после чего быстро строит работающий прототип.

Далее команда выделяет приоритетные высокоуровневые требования и определяет итерации проекта, включая требования и выходные ограничения для каждой итерации, зная при этом, что план будет изменяться. Они начинают первую итерацию, фиксируя детальные требования, с использованием инструментов для спецификации продукта. Быстро производится работающий прототип с помощью инструментов моделирования бизнес-сущности и бизнес-процесса, поставленных командой разработчиков линейки продуктов, привязывая Web-методы к операциям над сущностями и принимая сгенерированные реализации по умолчанию. Пересмотрев прототип с заказчиком, они модифицируют требования и выполняют ревизию прототипа. Заказчик подтверждает, что новый прототип выглядит хорошо. На этом завершается первая итерация. Прошла одна неделя.

Команда пилотного проекта уточняет требования и затем настраивает архитектуру и конфигурацию развертывания.

Команда начинает вторую итерацию с фиксации дополнительных требований, применяя инструменты спецификации продуктов. Теперь они используют точку зрения "Класс сущности" для написания кода на C# для добавления бизнес-логики к сущностям, сгенерированным из моделей. Затем они используют точку зрения "Web-служба" для модификации архитектуры, применяя шаблон "Фасад" (Façade). После каждого цикла изменений они тестируют рабочее программное обеспечение, дабы убедиться, что все идет хорошо, и исправляют дефекты, выявленные в процессе тестирования. Далее они специфицируют настраиваемую политику безопасности, требуемую заказчиком, как аспект, и применяют ее ко всем Web-методам фасада. Затем обновляют конфигурацию развертывания для включения установок, продиктованных настраиваемой политикой безопасности, и снова тестируют программное обеспечение, чтобы убедиться, что все хорошо. После исправления еще нескольких дефектов в их написанной вручную бизнес-логике, они получают работающее приложение, готовое к показу заказчику. Это завершает вторую итерацию. Прошло еще две недели.

Команда пилотного проекта строит часть приложения, которая находится вне области действия фабрики.

Наконец, команда приступает к построению специализированной части приложения, которая не относится к юрисдикции фабрики программного обеспечения. В это время они вносят многочисленные изменения в фабричную часть приложения для подгонки к требованиям специализированной части, включая написание специальных обработчиков сообщений для фасада Web-службы. После длительной фазы исправления дефектов в специализированной части приложения пилотный проект завершается. От начала до конца он занимает три месяца. Обратите внимание, что фабричная часть приложения покрывает 75% функциональности, но занимает около 25% времени, в то время как специализированная часть имеет в точности противоположный профиль, поскольку полностью кодируется вручную с нуля. Заказчик в экстазе, и не-

медленно просит внести изменения. Поскольку определение приложения зафиксировано — от требований, через конфигурацию развертывания — затраты и реализация этих изменений проходят гораздо легче, чем это было бы без фабрики ПО.

Мы надеемся, что этот простой пример дает почувствовать подход на основе фабрик ПО. Очевидно, что многие детали вроде оптимизации производительности, того, как изменчивость требований отражается на изменчивости архитектуры, реализации, конфигурации развертывания, плане тестирования и других частях жизненного цикла программного обеспечения, опущены. В главе 16 мы рассмотрим подробный пример, иллюстрирующий все упомянутые вещи.

Более детализированный пример представлен в главе 6.

Последствия фабрик программного обеспечения

Фабрики программного обеспечения продвигают разработку ПО в сторону индустриализации. Как было сказано в предисловии, одной из наших целей является демонстрация того, как новые технологии могут быть использованы для снижения затрат за счет внедрения автоматизации, обеспечивая при этом выигрыш в производительности, и не только в широких горизонтальных доменах, подобных конструированию пользовательского интерфейса и доступу к данным, но также в узких вертикальных доменах, таких как здравоохранение и финансовые услуги. В этом разделе мы рассмотрим влияние фабрик ПО и опишем, как может выглядеть отрасль, когда эти фабрики станут широко распространенными. Рисуя эту картину, мы неизбежно пропустим многие сложности. Это не значит, что их можно игнорировать. Значительную часть книги мы посвятили их идентификации и предполагаемым стратегиям их разрешения. Но поскольку это только изображение, оно достаточно близко к реальности, чтобы направить ход наших мыслей.

Мы рассмотрим, как изменится отрасль по мере распространения фабрик ПО.

Разработка сборкой

Разработчики приложений будут строить только малую часть каждого приложения — обычно меньше трети. Все остальное составят существующие компоненты[2]. Разработка приложений будет состоять в основном из настройки, адаптации, расширения и сборки компонентов. Вместо написания огромного объема нового кода, большую часть нужной функциональности они получат от готовых и построенных на заказ компонентов, поставляемых по контракту сторонними поставщиками, каждый из которых будет делать то же самое.

Только небольшая часть каждого приложения будет разрабатываться с нуля.

[2] Хотя эти количества могут показаться чрезмерными, многие случаи доказали, что даже более высокие уровни повторного использования могут быть достигнуты и поддерживаться благодаря применению линеек программных продуктов [48, 26, 245].

Цепочки поставщиков программного обеспечения

В ответ на потребность в компонентах возникнут цепочки поставщиков.

Цепочки поставщиков возникнут, чтобы удовлетворить потребность в компонентах. Согласно Ли (Lee) и Биллингтону (Billington) [146], цепочка поставщиков — это сеть, которая начинает с сырых материалов, трансформирует их в товары-полуфабрикаты, а затем — в конечные продукты, поставляемые заказчикам через систему дистрибуции. Каждый участник получает логические и/или физические продукты от одного или более вышестоящих поставщиков, добавляет им стоимость, обычно включая их в более сложные продукты, и передавая результат нижестоящим потребителям.

Форматы спецификаций, форматы упаковки, архитектуры и шаблоны будут стандартизованы.

Продукты, разработанные цепочкой поставщиков, охватывает множество организаций, отделяя потребителей от поставщиков. Это создает потребность в стандартизации перечисленных ниже вещей.

- Форматы спецификации продуктов — для упрощения согласований между потребителями и поставщиками.
- Активы распространенных доменов — особенно архитектур и шаблонов — для облегчения сборки независимо разработанных компонентов с пересечением границ платформ с предсказуемым результатом.
- Форматов пакетирования, содержащие метаданные компонентов, облегчающие поиск, выбор, лицензирование, конфигурирование, инсталляцию, адаптацию, сборку, развертывание, мониторинг и управление компонентами.

Компоненты общего употребления станут товаром, и появятся поставщики, специализирующиеся на широком разнообразии доменов, делая программное обеспечение, слишком дорогое при разработке современными методами, легко доступным.

Как формируются цепочки поставщиков

Фабрики программного обеспечения могут быть разделены на части, формируя цепочки поставщиков. Одним из способов достижения этого может быть получение компонентов от вышестоящих поставщиков.

Фабрики ПО могут быть разделены на части — как вертикально, так и горизонтально, — чтобы передать ответственность внешним поставщикам, порождая, таким образом, цепочки поставщиков.

- Вертикальное разделение позволяет фабрике ПО собирать компоненты, полученные от вышестоящих поставщиков. Представьте, например, что каркас сущностей из предыдущего примера поступил от компании — независимого разработчика ПО (Independent Software Vendor — ISV).
- Горизонтальное разделение отделяет разработчиков линейки продуктов и разработчиков продукта на одном уровне цепочки поставок. Это может принимать одну из двух форм:
  - Разработка линейки продуктов передается внешним разработчикам (аутсорсинг). Предположим, например, что вместо работы для ISV в предыдущем примере, разработчики линейки продуктов, которые создают фабрику ПО, работают на внешнего системного интегратора (Systems Integrator — SI). Вместо построения фабрик ПО для собственных разработчиков, они создают их для разработчиков организации-заказчика.

Другой способ разделения фабрики — передача разработки продукта или линейки продуктов внешним разработчикам (аутсорсинг).

- Разработка продукта передается внешним разработчикам (аутсорсинг). Предположим, например, что разработчики продукта в предыдущем примере работают на SI, используя фабрику ПО, созданную разработчиками линейки продуктов, работающими на ISV. Разработчики продуктов могут находиться за границей — в странах с менее дорогостоящим рынком труда.

Конечно, оба типа разделения могут возникать в любом месте фабрики и могут комбинироваться произвольным образом. Они также могут появляться и исчезать, как диктуют условия бизнеса. В зрелых отраслях множество уровней поставщиков вносят свой вклад в создание финального продукта, причем происходит постоянная ротация — поставщики все время вступают в отношения и покидают их.

Эти типы разделения могут комбинироваться.

Управление отношениями

Управление отношениями с заказчиками и партнерами будет становиться все более важным. Соглашения уровня служб будут регламентировать транзакции между поставщиками и потребителями. Обслуживание и помощь будут сопровождаться гарантиями. Потребители будут арендовать компоненты у поставщиков и систематически получать заплаты и обновления. Механизмы динамического обновления станут менее навязчивыми. Инструменты, управляющие конфигурациями развернутых продуктов, станут важнейшей частью платформы. Данные о взаимодействии с заказчиками и партнерами будут накапливаться и анализироваться для повышения уровня обслуживания, для оптимизации производства и поставок и для планирования предложений продуктов.

Важность управления отношениями с заказчиками и партнерами будет возрастать.

Активы, специфичные для домена

Разработчики продуктов станут редко применять языки программирования общего назначения, за исключением лишь небольших частей продукта. Вместо этого они станут использовать инструменты, подобные построителям пользовательских интерфейсов, для сборки компонентов на базе каркасов, применяя специфичные для домена языки. Эти языки и инструменты, недорогие в производстве, станут создавать организации, обладающие знаниями домена предметной области, предлагая экономически выгодные решения для таких вертикальных доменов, как банковское дело или здравоохранение. Разработчики линейки продуктов часто станут прменять языки программирования общего назначения для построения производственных активов, используемых разработчиками продуктов — почти так же, как в наши дни разработчики операционных систем создают драйверы устройств, а разработчики другого системного программного обеспечения используют их.

Разработчики линейки продуктов будут создавать активы, используемые разработчиками продуктов.

Организационные изменения

Индустриализация затронет всех и каждого, кто связан с разработкой программного обеспечения. Разработчики разделятся на большее число специализированных ролей — по мере того, как процессы будут все более стандартизованы и автоматизиро-

Многое изменится в разработке и ее организации.

ваны. Эта эволюция приведет к стандартизации ролей и профессиональной организации, которые будут больше походить на те, что существуют в других областях. Разработчики и архитекторы станут лицензированными специалистами, а некоторые стандарты станут обязательными к исполнению. В наше время уже существует тенденция к лицензированию специалистов-практиков и идентификации знаний, необходимых для компетентной практики. Конечно, сертификация, свидетельствующая, что разработчики знакомы с определенными фактами, не гарантирует, что они могут строить программное обеспечение, как знание правил бухгалтерии не гарантирует, что бухгалтер сможет идеально вести дела. Начали появляться независимые центры оценки квалификации. Сертификация в таких центрах будет требовать представления и защиты работ, выполненных соискателями в данной области. Поставщики получат квалификацию и мотивировку для построения фабрик ПО и для участия в цепочках поставщиков. Они реструктуризируют существующие продукты в семейства. Потребители получат ускорение возврата средств, высокое качества и низкие затраты — и все это за счет индустриализации.

Массовая подгонка программного обеспечения

Со временем программное обеспечение может стать массово настраиваемым, как персональные компьютеры, заказываемые через Web сегодня.

Некоторые отрасли, такие как бизнес персональных компьютеров в Web, быстро и дешево производят сегодня варианты продуктов по заказу индивидуальных потребителей. Поскольку в долговременной перспективе можно ожидать потребности в массовой подгонке программных продуктов по заказу потребителей, фабрики программного обеспечения должны обеспечить такую возможность. Потребность в массовой подгонке возникает в наши дни и в других областях, и она обеспечивается путем оптимизации внутренних цепочек возрастания стоимости компаний-поставщиков. Согласно Портеру (Porter), цепочка повышения стоимости — это последовательность действий, добавляющих стоимость продукта и происходящих внутри отдельной организации, соединяющая сторону поставки со стороной спроса. Оптимизация цепочки возрастания стоимости требует интеграции бизнес-процессов, подобных управлению отношениями с заказчиками (CRM), управлению спросом, определению продукта, дизайну продукта, сборке продукта и управлению цепью поставщиков, как показано на рис. 5.12. Массовая подгонка программных продуктов невозможна до тех пор, пока поставщики не оптимизируют собственные цепочки приращения стоимости. Представьте себе заказ настраиваемого приложения финансовой службы через Web-сайт — точно так же, как сегодня заказываются персональные компьютеры в специальной конфигурации, — но с длительным процессом формирования заказа, состоящего из намного большего числа шагов, однако с тем же уровнем конфиденциальности и сравнимыми сроками поставки.

Реализация видения фабрики программного обеспечения

Это — видение следующего десятилетия, но оно уже начало материализовываться.

Реализация этого видения потребует от нас изменения образа мышления в отношении экономики разработки программного обеспечения. Например, организации, обладающие знаниями проблемной области, такие как “полевые” организации, неза-

висимые поставщики программ (ISV), системные интеграторы (SI) и информационные службы предприятий, зафиксируют свои знания в языках, каркасах, шаблонах и инструментах. Вместо производства "сырых" артефактов знаний вроде чертежей и документов, носителями знаний станут поставщики инструментов, процессов и содержимого — то есть участники бизнеса, который в наши дни экономически выгоден только для небольшого числа поставщиков. Эти новые активы автоматизируют разработку семейств продуктов в специфических доменах, позволяя пользователям собирать члены этих семейств из общих компонентов, используя комбинации стандартных шаблонов.

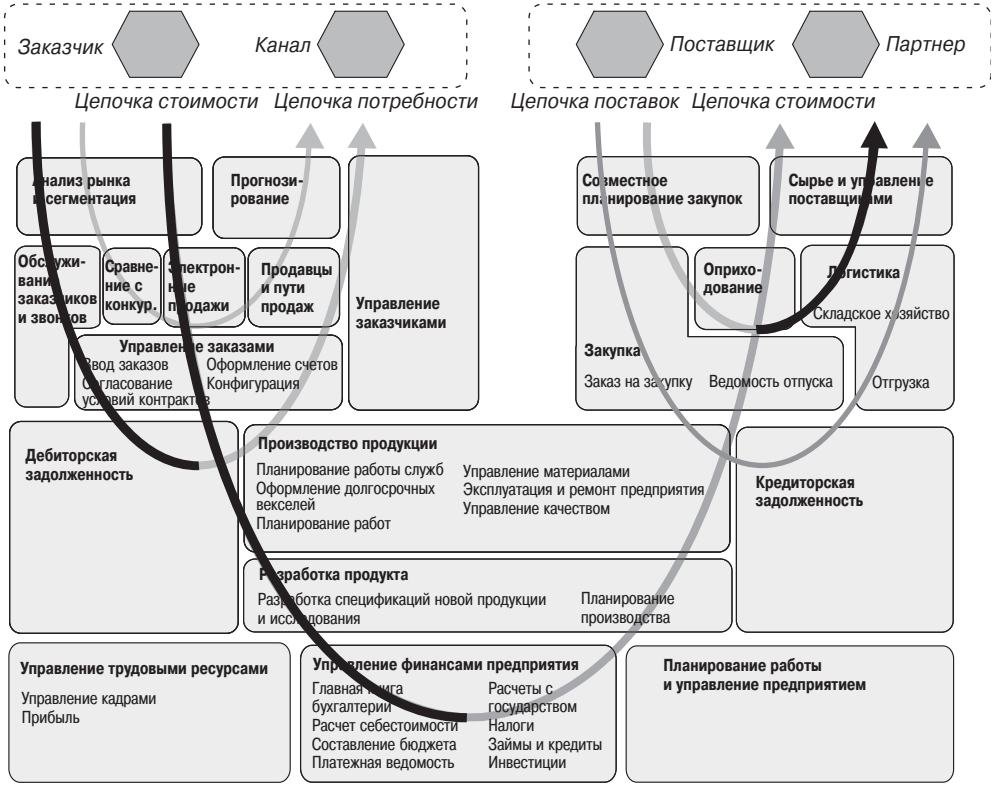

Рис. 5.12. *Оптимизация внутренней цепочки приращения стоимости*

Фабрики программного обеспечения собирают все эти идеи вместе.

Фабрики ПО основаны на сплаве ключевых идей о систематическом повторном использовании, разработке сборкой, управляемой моделями разработке и каркасы процессов. Многие из этих идей не новы. Новым является их синтез в интегрированный и в большей степени автоматизированный подход, позволяющий организациям со знаниями предметной области реализовать шаблон фабрики ПО из четырех частей, строить языки, шаблоны, каркасы и инструменты для автоматизации разработки в узких доменах. Уже было сказано, что мы переоцениваем то, что может быть сделано за 5 лет, но недооцениваем то, что может быть сделано за 10 лет.

Мы полагаем, что ключевые части представления о фабриках ПО будут реализованы, некоторые быстро, а некоторые — на протяжении следующего десятилетия. Коммерческие инструменты, обслуживающие фабрики ПО, доступны уже сейчас, включая Microsoft Visual Studio и IBM WebSphere® Studio. Технология DSL намного новее, чем большинство других, и основана на семействе расширяемых языков, о чем будет сказано в главе 9. Однако инструменты разработки DSL и каркасы в настоящее время находятся в процессе развития (например, проект GME [142]), и уже начали появляться в коммерческом виде.

Следующая, вторая, часть книги посвящена более подробному описанию критических инноваций.

В части II.