Содержание

Часть 2. Технологии реализации

Часть 3. Практические примеры

3 Инженерия предметной области и объектно-ориентированные методы анализа и проектирования

3.1. Почему эту главу стоит прочесть

За счет значительных преимуществ по сравнению со структурными методами, объектно-ориентированные методы анализа и проектирования нашли в нашей индустрии широкое применение. К сожалению, они не совсем приспособлены к обеспечению повторного использования программного обеспечения. В частности, они не подходят для разработки каркасов и компонентов. Решением этой проблемы представляется интеграция инженерии предметной области и объектно-ориентированной технологии. В результате мы получим высокопроизводительные методики объектно-ориентированного моделирования, встроенные в среду разработки семейств систем.

В этой главе мы поговорим о недостатках объектно-ориентированной технологии, имеющих отношение к разработке программного обеспечения многократного применения, а также о способах объединения инженерии предметной области и объектно-ориентированных методов, способных снять эти недостатки. Кроме того, мы представим обзор таких интегрированных методов.

3.2. Объектно-ориентированная технология и возможность повторного использования

Когда объектно-ориентированная технология только появилась, бытовало мнение о том, что сам характер объектов предполагает возможность повторного использования, а значит, программные средства многократного применения как побочный продукт разработки приложений «рождаются» сами собой (такая мысль, к примеру, прослеживается в работе [HC 91]). Сегодня сообщество специалистов

в области объектно-ориентированной технологии уже прекрасно понимает, насколько это заявление далеко от правды. Объектно-ориентированное программное обеспечение многократного применения требует тщательной разработки, а разработка *для повторного использования* (for reuse) предполагает значительные капиталовложения (см., например, [GR95]).

Так каким же образом возможность повторного использования реализована в современной объектно-ориентированной технологии? По мнению Коэна (Cohen) и Нортропа (Northrop) [CN98], на этот вопрос нужно отвечать с двух позиций: пространства задачи (то есть методов анализа) и пространства решений (то есть технологий реализации).

3.2.1. Пространство решений

В рамках объектно-ориентированной технологии есть две области, связанные с пространством решений и направленные на многосистемное проектирование: это каркасы и образцы проектирования. В состав каркаса входит абстрактная проектная схема семейства родственных систем, представленная в виде сотрудничающих классов. Образцы проектирования, в свою очередь, являют собой многократно используемые решения регулярно встречающихся в различных системах проектных проблем. Кроме того, образцы, как разновидность документации, помогают фиксировать многократно используемые решения в других областях — например, в области анализа [Fow97], архитектуры [BMR+96] и в организационных вопросах [Cop95].

К сожалению, все это не снимает две основных проблемы.

- *Объектно-ориентированные методы анализа и проектирования* (OO Analysis and Design, OOA/D), способные обеспечить возможность разработки каркасов, крайне немногочисленны. Систематизированные средства обеспечения поиска и применения образцов также почти отсутствуют. Поскольку эти вопросы связаны с анализом пространства задачи, мы рассмотрим их в следующем разделе.
- Самая насущная проблема современной каркасной технологии заключается в том, что по мере универсализации каркасов ощущается все возрастающее усложнение и снижение производительности. К примеру, задействовав классические проектные модели из [GHJV95], мы можем оснастить каркас новыми изменяемыми параметрами. С другой стороны, обычно это приводит к излишней фрагментации конструкции и появлению «множества мелких методов и классов». Кроме того, каркасная технология сильно зависит от динамического связывания; оно осуществляется даже при реализации изменчивости между приложениями, хотя в этой ситуации разумнее применять статическое связывание и частичное вычисление (см. главу 10). В результате происходит неоправданное снижение производительности, а в конечных приложениях появляется неиспользуемый код. Помимо этого, современные объектно-ориентированные языки не позволяют нам должным образом отделять друг от друга и фиксировать такие важные аспекты, как синхронизация, удаленное взаимодействие, управление памятью, устойчивость и т. д. Все эти замечания справедливы и по отношению к компонентным технологиям — в частности, ActiveX и JavaBeans.

Для решения указанных проблем требуется комплексная методика, включающая новые лингвистические конструкции, новые механизмы композиции, возможность метапрограммирования и т. д. Все эти вопросы мы обсудим в главах 8 и 9.

3.2.2. Пространство задачи

Традиционные методы OOA/D, такие как OOSE [JCJO92], Booch [Boo94], OMT [RBP91] и даже текущая версия (5.0) рационального унифицированного процесса (Rational Unified Process) [JBR99], сориентированы на разработку одиночных систем и не приспособлены для создания семейств систем[1]. Учитывая, что мы преследуем последнюю из этих целей, перечисленные методы следует признать негодными для разработки программных средств многократного применения, ибо для этого требуется ориентация на группы систем, а не на отдельные системы.

В этом контексте можно привести следующие недостатки традиционных методов OOA/D, касающиеся их технологического процесса и нотаций моделирования.

- *Отсутствие различий между разработкой для повторного использования и разработкой с повторным использованием.* Для реализации принципа многократного применения требуется разбиение процесса разработки объектно-ориентированного программного обеспечения на разработку для повторного использования (то есть инженерию предметной области) и разработку с повторным использованием (прикладную инженерию). Областью разработки для повторного использования должно быть семейство систем; лишь в этом случае появляется возможность производства компонентов многократного применения. Процесс разработки с повторным использованием должен быть организован таким образом, чтобы в нем могли быть задействованы повторно используемые средства, созданные в ходе разработки для повторного использования. Современные процессы OOA/D не отличаются такими качествами. Методы OOA/D можно с некоторой натяжкой уподобить лишь прикладной инженерии со случайным (вместо систематического) употреблением повторно используемых средств.
- *Отсутствие этапа определения предметной области.* Поскольку методы OOA/D ориентированы на разработку одиночных систем, они не предусматривают определения предметной области — этапа, на котором производится отбор целевой группы систем. Кроме того, методы OOA/D направлены на удовлетворение запросов «того самого» покупателя одиночной системы; при этом анализ круга людей, заинтересованных в появлении данной группы систем (и включающего потенциальных покупателей), не производится.
- *Отсутствие различий между моделированием изменчивости в рамках одного приложения и между несколькими приложениями.* В современных объектно-ориентированных нотациях не проводится никаких различий между изменчивостью внутри приложения — к примеру, изменчивостью объектов во времени или применением различных вариантов объекта в разных местах

[1] Обзор методов OOA/D содержится в работе [Fow96].

приложения — и изменчивостью между приложениями, то есть изменчивостью, распространяющейся на различные приложения, предназначенные для различных пользователей и контекстов применения. Более того, те механизмы реализации, которые в объектно-ориентированном программировании предназначены для реализации изменчивости внутри приложений (например, динамический полиморфизм), задействуются и для обеспечения изменчивости между приложениями. Следствием такой практики является появление «жирных» компонентов и каркасов, из которых получаются «жирные» приложения.

- *Отсутствие средств моделирования изменчивости, независимых от реализации*. Помимо всего прочего, современные объектно-ориентированные нотации не поддерживают независимое от реализации моделирование изменчивости — скажем, составляя диаграмму классов UML, приходится решать, как лучше представить данный изменяемый параметр — путем наследования, агрегации, параметризации классов или при помощи какого-нибудь другого механизма реализации.

В отличие от традиционных методов OOA/D, некоторые недавно появившиеся методы — например, OOram [Ree92] и Catalysis [DW97] — обеспечивают полную поддержку моделирования каркасов и применения образцов проектирования. Что касается моделирования каркасов, важно, что в OOram фундаментальные абстракции объектно-ориентированных конструкций определяются не как классы, а как *кооперации* (collaborations). Кооперация состоит из нескольких *ролей* (roles), взаимодействие между которыми происходит по определенному образцу (паттерну). Конкретные объекты могут играть несколько ролей в одной или нескольких кооперациях. Таким образом, в качестве единственного средства компоновки классов выступает композиция кооперaций. Каркас тоже лучше моделировать как композицию кооперaций, но не классов — дело в том, что расширение каркаса обычно предполагает согласованное расширение нескольких классов.

На момент написания этих строк OOram был единственным методом OOA/D, в котором была признана необходимость в выделении специального процесса разработки для повторного использования. Этот метод также предусматривает этап определения предметной области, выполняемый путем анализа различных групп потребителей. Кроме того, в его рамках проводится анализ существующих систем. К сожалению, моделирование характеристик в нем отсутствует. Методу OOram будет посвящен раздел 3.6.2.

Как указывалось ранее, общей для всех методов OOA/D проблемой является неудовлетворительность моделирования изменчивости. Несмотря даже на то, что различные методики моделирования, используемые в методах OOA/D, все же поддерживают механизмы изменчивости (например, наследование и параметризацию на диаграммах объектов, составление диаграмм сотрудничества и т. д.), методы OOA/D не предусматривают абстрактную и компактную модель общности, изменчивости и зависимостей. Существует несколько причин, доказывающих разумность создания такой модели.

- Поскольку в разных моделях одинаковую изменчивость можно реализовать при помощи различных механизмов, нам требуется более абстрактное представление изменчивости (см. разделы 3.6.3 и 4.5).

- Пользователи программных средств многократного применения нуждаются в четком и кратком представлении имеющихся характеристик и изменчивости.
- Разработчик программных средств многократного применения обязан найти ответ на вопрос: «Почему в этом повторно используемом программном продукте присутствует данная характеристика или изменяемый параметр?»

Отсутствие этапа определение предметной области и четкого механизма моделирования изменчивости может привести к появлению двух серьезных проблем.

- Нехватка значимых характеристик и изменяемых параметров.
- Многие характеристики и изменяемые параметры присутствуют, но не используются; следовательно, повышается сложность и возрастают неоправданные издержки (стоимость разработки и сопровождения).

Для того чтобы добиться оптимального охвата характеристик и изменяемых параметров, необходимо соблюдать баланс между текущими и перспективными потребностями. Таким образом, нам требуется подробная модель с обобщением характеристик и изменяемых параметров, логическим обоснованием и учетом лиц, заинтересованных в каждом из них. В рамках инженерии предметной области эту роль исполняет характеристическая модель (см. разделы 2.7.4 и 4.4). В ней фиксируется все, что касается способности программных продуктов многократного применения к изменению конфигурации и повторному использованию.

3.3. Связь между инженерией предметной области и объектно-ориентированными методами анализа и проектирования (OOA/D)

В главе 2 мы пришли к выводу, что инженерия предметной области ориентирована на разработку решений, связанных с группами программных систем. Современные методы OOA/D, напротив, направлены на проектирование одиночных систем. Подобное расхождение в целях, как мы установили в предыдущем разделе, приводит к тому, что существующие на сегодняшний день методы OOA/D оказываются неприспособленными к разработке программных средств многократного применения.

В то время как инженерия предметной области обеспечивает поддержку мультисистемного проектирования и предусматривает отвечающие нашим требованиям методики моделирования изменчивости, методы OOA/D заключают в себе высокоэффективные методики системного моделирования. Из этого следует, что методы инженерии предметной области и OOA/D очень разумно объединить. В самом деле, именно такой интеграцией в наше время озабочено сообщество специалистов по инженерии предметной области (см., например, [CN98]).

3.4. Перспективы интеграции методов инженерии предметной области и OOA/D

В процессе интеграции методов разработки мы должны обращать внимание на отдельные объединяемые сферы.

- *Задачи методов.* В нашем примере методы инженерии предметной области направлены на обеспечение разработки моделей групп систем; методы OOA/D, напротив, ориентированы на одиночные системы.
- *Принципы.* В состав анализа предметной области входит не только моделирование понятий данной предметной области, но и выявление альтернативных методов реализации (это делается для того, чтобы определить надлежащую терминологию и рамки последующих этапов); OOA, наоборот, не связан с вопросами реализации.
- *Процессы.* Инженерия предметной области включает в себя разработку *для* повторного использования, а прикладная инженерия — разработку *с* повторным использованием; в то же время в большинстве методов OOA/D никаких различий между этими понятиями не проводится.
- *Модели и нотации.* Инженерия предметной области предусматривает новые виды моделей — например, характеристические модели; методы OOA/D обеспечивают все необходимые методики системного моделирования.

Поскольку существующие на сегодняшний день методы отличаются высокой сложностью, их интеграция зачастую также оказывается очень непростой задачей. Кроме того, в силу того что тестирование и усовершенствование интегрированных методов возможно только в ходе их применения в рамках реальных проектов, длительность этого процесса и вложения в него могут быть очень значительными. (Общие вопросы интеграции методов рассматриваются в работе [Son97].)

Так как в настоящее время методы OOA/D используются шире, чем инженерия предметной области, мы начнем с обзора тех изменений, которые в них следует внести.

- *Корректировка процессов.* Здесь в первую очередь необходимо ввести отдельные процессы разработки для повторного использования (инженерии предметной области) и разработки с повторным использованием (прикладной инженерии). Структура процесса инженерии предметной области может быть довольно сложной: среди его компонентов могут оказаться процесс прикладной инженерии семейства приложений, а также несколько горизонтальных процессов инженерии предметной области (см. раздел 3.5). За исключением, в первую очередь, того, что процесс прикладной инженерии ориентирован на разработку решений на основе имеющихся повторно используемых средств, он зачастую демонстрирует значительное сходство с традиционным процессом OOA/D. С другой стороны, в процессы инженерии предметной области следует дополнительно ввести действия по определению предметной области и моделированию характеристик.

- *Моделирование изменчивости и зависимостей.* Моделирование изменчивости и зависимостей — это один из основных элементов инженерии предметной области. На разных стадиях процесса разработки представление изменчивости осуществляется в разных моделях. Как правило, моделирование изменчивости начинается на таксономическом уровне с разработки словаря с описанием различных понятий. К примеру, обычно, прежде чем построить объектную модель банковских счетов, мы сначала обсуждаем различные их типы (например, срочные счета и счета до востребования). Характеристические модели позволяют нам зафиксировать этот таксономический уровень и заготовить справочную информацию по изменчивости в рамках других моделей (например, объектных моделей, моделей Use Case, моделей взаимодействия, моделей переходов и т. д.). Их совершенно необходимо присовокупить к тому набору моделей, который используется в объектно-ориентированной разработке программных средств в настоящее время. Не меньшее значение представляет фиксация зависимостей между характеристиками. Она позволяет нам проводить автоматическое конфигурирование (например, конфигурирование на основе ограничений), избавляющее последующих пользователей от некоторых конфигурационных манипуляций, выполняемых вручную.
- *Разработка инфраструктуры повторного использования.* В дополнение к разработке повторно используемых средств, мы должны спроектировать и ввести в действие инфраструктуру повторного использования, с помощью которой можно будет упаковывать, хранить, распространять, искать, оценивать и объединять эти средства.

Мы уже упоминали о том, что сообщество специалистов по инженерии предметной области в настоящее время серьезно занимается проблемой интеграции своего профильного метода и методов OOA/D. Действия, направленные на такую интеграцию, можно подразделить на четыре категории.

- *Обновление устаревших методов инженерии предметной области.* В старых методах инженерии предметной области наподобие FODA (см. раздел 2.8.1) в качестве методики разработки систем применялись структурный анализ и проектирование. Сегодня ведется работа по замене этих устаревших методик новыми приемами OOA/D; в качестве примера здесь можно привести разработку FODAcom [VAM+98] — новой, объектно-ориентированной версии FODA, предназначенной специально для предметной области телекоммуникаций.
- *Конкретизация настраиваемых методов инженерии предметной области.* В новых методах инженерии предметной области наподобие ODM (см. раздел 2.8.2) методики системного проектирования рассматриваются как параметры. Следовательно, прежде чем задействовать параметризованный метод инженерии предметной области, его необходимо конкретизировать до какого-либо метода конструирования конкретных систем. Осуществление подобной конкретизации может, впрочем, оказаться довольно непростой задачей. Как указано в работе [Sim97], метод ODM конкретизирован на взаимодействие с методом OOA/D под названием Fusion [CAB+94]. В качестве другого примера выполнения этой задачи можно упомянуть метод конструирования алгоритмических библиотек многократного применения на основе инженерии

предметной области (Domain Engineering Method for Reusable Algorithmic Libraries, DEMRAL), рассматриваемый в главе 5[1].

- *Расширение существующих методов OOA/D*. Третий подход предполагает дополнение одного из существующих методов OOA/D понятиями из инженерии предметной области. Примером подобных действий может послужить инициатива по разработке программного обеспечения на основе повторного использования (Reuse-driven software Engineering Business, RSEB) [JGJ97], использующая объектно-ориентированную нотацию моделирования UML [Rat98c] и объектно-ориентированный метод разработки программных средств (OO Software Engineering, OOSE) [JCJO92]. Как вы уже знаете, для модернизации и адаптации к инженерии предметной области любого метода объектно-ориентированного системного конструирования требуется внести существенные изменения в структуру его технологического процесса. Кроме того, для того чтобы обеспечить возможность моделирования изменчивости, требуется расширить нотации моделирования (к примеру, внедрить диаграммы характеристик и изменяемые параметры).
- *Интеграция второго поколения*. Наконец, примером интеграции двух методов — FODAcom и RSEB (они упоминались выше), — в результате которой стало возможным сочетание понятий инженерии предметной области и OOA/D, является FeatuRSEB [GFA98]. Одним из недостатков первоначального описания RSEB было отсутствие возможности моделирования изменчивости при помощи характеристических моделей. В результате объединения этого метода с понятиями FODAcom эта проблема была снята. С другой стороны, RSEB в большей степени, чем FODAcom, носит объектно-ориентированный характер. Таким образом, от интеграции выиграли оба метода.

Различные инициативы, направленные на интеграцию, мы рассмотрим в разделе 3.6.

3.5. Горизонтальные и вертикальные методы

Мы уже успели представить понятия вертикальных и горизонтальных предметных областей (см. раздел 2.7.2). В состав горизонтальных предметных областей входит только одна часть системы — к примеру, графические пользовательские интерфейсы, системы управления базами данных, промежуточное программное обеспечение, библиотеки расчета матриц, библиотеки контейнеров, каркасы финансовых объектов и т. д. Вертикальные области, напротив, распространяются на целые системы — например, системы бронирования авиабилетов, медицинские информационные системы, системы автоматизированного проектирования и т. д. Очевидно, что для разных предметных областей подходят разные методы инженерии предметной области. Организации, специализирующейся в одной

[1] Следует заметить, что DEMRAL — это не просто конкретизация ODM; в этом методе объединены такие новаторские понятия, как наборы начальных характеристик и предметно-ориентированные языки конфигурирования.

или нескольких горизонтальных предметных областях, имеет смысл пользоваться методами, приспособленными именно для этих областей. Такие методы инженерии предметной области мы называем *горизонтальными*. Образцом горизонтального метода инженерии предметной области является DEMRAL (см. раздел 3.6.5). Что касается вертикальных предметных областей, то здесь от нас требуется разработать и впоследствии обслуживать общую архитектуру многократного применения, охватывающую систему в целом, а затем, при разработке повторно используемых моделей для подсистем, задействовать специализированные горизонтальные методы. Метод инженерии предметной области, распространяющийся на вертикальную предметную область, мы называем *вертикальным*. Пример вертикального метода конструирования — RSEB, рассматриваемый в разделе 3.6.3. Как правило, наша цель заключается в разработке модульных методов инженерии предметной области, которые можно приспособить к специфическим потребностям различных организаций. Этого можно достичь, если заставить вертикальный метод «вызывать» различные специализированные горизонтальные методы, относящиеся к разным подсистемам.

Между предметными областями, а значит, и между их методами инженерии предметной области, могут существовать существенные различия. Одним из основных дифференцирующих факторов является стиль моделирования.

- *Стиль взаимодействия*. В основном, стиль взаимодействия выражается во взаимодействии между объектами — в частности, между компонентами, схемами вызова процедур, потоками сообщений и событий. Моделирование взаимодействия производится при помощи элементов Use Case, кооперации и диаграмм последовательностей.
- *Алгоритмический стиль*. Основным признаком алгоритмического стиля являются алгоритмы, которые производят сложные операции вычисления в отношении абстрактных типов данных. Задание алгоритмов производится при помощи псевдокода или специальных нотаций спецификации.
- *Информационно-ориентированный стиль*. Такой стиль определяется, в первую очередь, структурой данных — к примеру, в моделировании баз данных. Моделирование структуры данных производится при помощи диаграмм объектов или отношений сущность—связь.
- *Потоковый стиль*. Основным аспектом потокового стиля является поток данных — например, в архитектурах каналов и фильтров при обработке сигналов. Задание потока данных производится при помощи диаграмм потоков данных.

Иногда нам, естественно, приходится задействовать все эти основные аспекты — взаимодействие, алгоритмы, структуры и поток данных — даже при работе над отдельной частью системы. Впрочем, очень часто ведущую, объединяющую роль исполняет какой-то один из них. К примеру, интерактивный характер большинства бизнес-приложений объясняется тем, что они обычно снабжаются интерактивным графическим пользовательским интерфейсом и структурируются как взаимодействующие компоненты в рамках распределенной, открытой среды. В крупных инженерно-технических системах (например, в системах автоматизированного проектирования) аспект взаимодействия также занимает главенствующие позиции. Его существенная роль во всех крупных системах объясняется

тем, что интеграция их подсистем производится именно посредством взаимодействия, которое реализуется через вызовы процедур, передачу сообщений, уведомления о событиях и т. д. По этой же причине большинство современных методов OOA/D относятся к элементам Use Case и являются сценарно-ориентированными; к примеру, этими качествами обладает рациональный унифицированный процесс (см. раздел 3.6.1). Вертикальный метод инженерии предметной области под названием RSEB (см. раздел 3.6.3), предназначенный для разработки крупных систем, также ориентирован на элементы Use Case. Впрочем, некоторые специализированные горизонтальные методы инженерии предметной области могут быть не направленными на элементы Use Case — к таким, в частности, относится DEMRAL, ориентированный на алгоритмические библиотеки. Как указано выше, вертикальный метод инженерии предметной области вызывает конкретизированные горизонтальные методы конструирования. Таким образом, вполне возможны ситуации применения элементов Use Case на системном уровне и отсутствия этой практики на некоторых уровнях подсистем.

ПРИМЕЧАНИЕ

Принимая во внимание, что и взаимодействие, и алгоритмы выражают поведение, читателю, вероятно, будет интересно узнать, в чем заключается разница между стилем взаимодействия и алгоритмическим стилем. Ее лучше всего проиллюстрировать путем сравнения Smalltalk и C++. Согласно основной метафоре Smalltalk, объекты трактуются как взаимодействующие агенты. С другой стороны, в C++ методы и функции рассматриваются скорее как математические операции над абстрактными типами данных. Эти два мнения вполне совместимы друг с другом, то есть мы можем пользоваться метафорой взаимодействия в C++ и метафорой операции в Smalltalk. В то же время отношение к одному из этих мнений как к доминирующему способно скорректировать многие конкретные проектные решения в обоих этих языках, в том или ином направлении. Любопытно, что Вегнер (Wegner) [Weg97, Weg98] рассматривает взаимодействие как новое качество объектно-ориентированной парадигмы. Он утверждает, что традиционная машина Тьюринга, способная изображать алгоритмы, не может должным образом фиксировать взаимодействия. В качестве новых аспектов взаимодействия выступают время и открытость. В силу своей открытости система может получать от окружения неалгоритмические стимулирующие воздействия, и, соответственно, ее поведение также принимает неалгоритмический характер (в таком случае его невозможно кодировать при помощи традиционных машин Тьюринга).

Более того, у каждой предметной области могут быть особые свойства, предполагающие применение специальных методик моделирования — к примеру, поддержка работы в реальном времени, распределение, параллелизм, устойчивость и т. д. Следовательно, в методах инженерии предметной области должна быть предусмотрена поддержка различных, специализированных на таких аспектах методик моделирования.

3.6. Избранные методы

Теперь мы рассмотрим несколько методов OOA/D, которые, за исключением первого — рационального унифицированного процесса версии 5.0, — содержат те или иные качества метода инженерии предметной области. Унифицированный процесс в данном случае приводится как пример известного объектно-

ориентированного метода, не приспособленного для разработки программных продуктов многократного применения.

3.6.1. Рациональный унифицированный процесс 5.0

Рациональный унифицированный процесс (Rational Unified Process)[1] версии 5.0 является неофициальным стандартом объектно-ориентированной разработки программных систем, популяризацией которого занимается компания Rational Software. Прообразом этого процесса стал метод объектно-ориентированной разработки программного обеспечения, описанный Джекобсоном (Jacobson) в соавторстве с другими исследователями [JCJO92] и впоследствии превратившийся в рациональный процесс Objectory (Rational Objectory Process) [Rat98a, Rat98b]. Задача унифицированного процесса заключается в том, чтобы «обеспечить изготовление высококачественного программного продукта, отвечающего требованиям конечных пользователей, не выходя за рамки временных и бюджетных ограничений». Унифицированный процесс *итеративен* (iterative) и *ориентирован на варианты использования* (use-case-centric); наличие этих двух качеств является необходимым условием успешной разработки масштабных программных систем. Варианты использования (элементы Use Case) выполняют функцию интеграции всей конструируемой системы на всех этапах и во всех моделях разработки. Этот принцип реализован в модели представлений программной архитектуры «4+1», созданной Крюхтеном (Kruchten) (рис. 3.1).

Унифицированный процесс организуется в двух измерениях (рис. 3.2).

- *Временное измерение*. Отражает прохождение стадий жизненного цикла с течением времени.
- *Измерение компонентов процесса*. Логическая систематизация действий по их характеру.

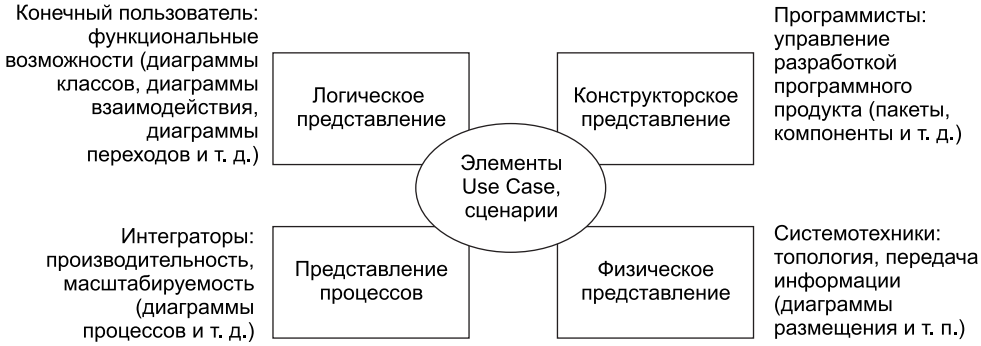

Рис. 3.1. Модель представлений «4+1» архитектуры преимущественно программной системы (видоизмененный вариант схемы, опубликованной в работе [Kru95], © 1995 IEEE)

[1] Достаточно часто название этого процесса переводят как «унифицированный процесс компании Rational», подчеркивая, что процесс разработан в упомянутой фирме. — *Примеч. научн. ред.*

Временное измерение подразделяется на циклы, этапы и итерации; друг от друга их отделяют контрольные точки. Циклом называется полное прохождение всех этапов. Компоненты процессов описываются в категориях действий, рабочих потоков, структурирующих эти действия, созданных артефактов и исполнителей.

Унифицированный процесс по определению ориентирован на разработку одиночных систем. По причине свойственных ему недостатков, которые мы рассмотрели в предшествующих разделах, он не годится для проектирования программных продуктов многократного применения.

- В нем не проводится никаких различий между разработкой для повторного использования и разработкой с повторным использованием.
- Он не предусматривает определение предметной области и мультисистемный анализ заинтересованных лиц.
- В нем отсутствуют операции, связанные с анализом характеристик.
- Он не содержит характеристических моделей.

ПРИМЕЧАНИЕ

Вариант использования (элемент Use Case) — это группа сценариев, описывающих определенное применение системы — например, снятие денежных средств в системе банковского обслуживания. Понятие элемента Use Case впервые появилось в работе [JCJO92]. За более подробными сведениями об элементах Use Case обращайтесь к изданиям [JBR99] и [EP98].

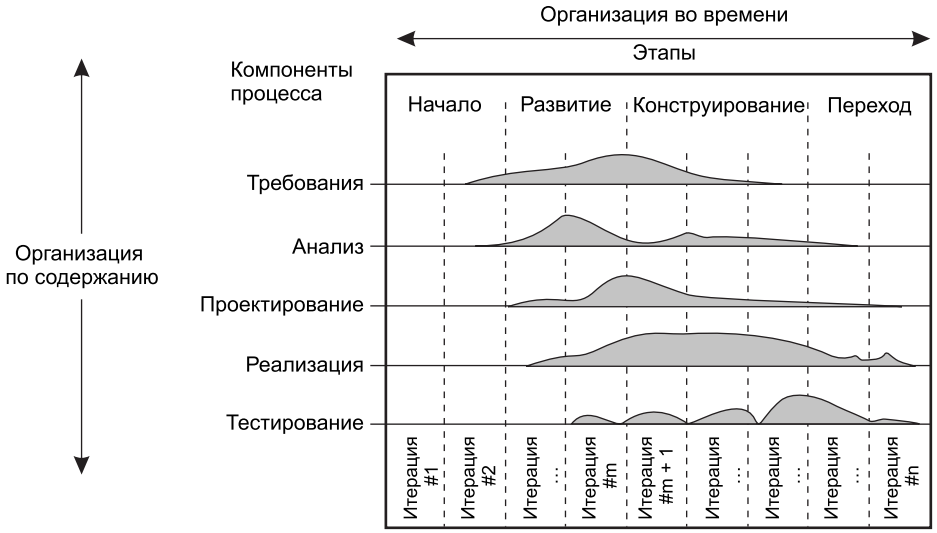

Рис. 3.2. Этапы и компоненты унифицированного процесса (видоизмененный вариант схемы, опубликованной в работе [JBR99])

3.6.2. OOram

OOram [Ree96] — это типовая система для создания разнообразных объектно-ориентированных методик, основанных на моделировании ролей. Разработчиком OOram, появившейся в еще 1970-х годах, является норвежец Трюгве Реенскауг

(Trygve Reenskaug). К примеру, понятия, реализованные в ранней версии OOram, были задействованы в парадигме «модель-представление-контроллер» Smalltalk, которая разрабатывалась силами Гольдберга (Goldberg) и Реенскауга в исследовательском центре Xerox PARC с 1979 по 1980 год [Ree96].

Базовым принципом OOram является *моделирование ролей* (role modeling). Основное внимание при моделировании ролей сосредоточено не на классах объектов, а на их *кооперации* (collaborations), а также на ролях, исполняемых этими объектами в рамках кооперации. Классы конструируются исключительно путем композиции ролей, исполняемых их экземплярами в различных кооперациях. Ролевая модель выражается через ряд представлений, из которых отдельного упоминания заслуживает *кооперативное представление* (collaboration view), отражающее отношения между ролями, и сценарное представление (scenario view), фокус которого направлен на взаимодействия. Для изображения этих представлений подходит нотация UML, хотя в оригинальном описании [Ree96] OOram приводится оригинальная нотация. В частности, кооперативное представление выражается через диаграммы сотрудничества, а сценарное представление — через диаграмму последовательностей.

Анализ сценариев предусматривается и в других объектно-ориентированных методах (взять тот же унифицированный процесс), однако они сориентированы на порождение из сценариев классов конкретных объектов (например, сначала создают элементы Use Case, затем на их основе получаются диаграммы сотрудничества и последовательности, и, наконец, производится определение классов). В OOram внимание сосредоточено на композиции кооперации. При необходимости к существующим композициям можно добавить новые кооперации; при этом производится соответствующее обновление участвующих классов. Таким образом, в OOram первичными стандартными блоками являются не классы, а кооперации.

Поскольку каркасные конструкции можно без труда представить в виде композиций кооперации, одной из возможных целевых технологий OOram являются именно каркасы. Кроме того, прослеживаются устойчивые аналогии между моделированием ролей и образцами проектирования — некоторые кооперации представляют собой частные случаи последних. Взаимосвязь между моделированием ролей, каркасами и образцами проектирования в настоящее время является предметом многочисленных исследований (см., например, [Rie97]). В разделе 9.8.2.7 мы еще вернемся к этой теме.

В OOram отсутствуют подробные и конкретные предписания относительно технологического процесса; это сделано умышленно. В силу того, что каждый отдельный процесс необходимо приспосабливать к конкретному сочетанию продуктов, задействованному персоналу и рабочим условиям, OOram определяет лишь руководящие принципы и подкрепляет их рядом всесторонне проанализированных конкретных примеров. В работе [Ree96] Реенскауг и его соавторы отмечают, что «многие менеджеры мечтают о появлении такого технологического процесса, который смог бы обеспечить удовлетворительные решения для всех проектов сразу. Мы считаем, что эти мечты не только напрасны, но и в некоторых случаях опасны».

В OOram выделяются три типа процессов [Ree96].

1. «*Процесс создания модели* нацелен на выработку модели или какого-либо другого средства выражения мыслей по поводу того или иного явления. В качестве примеров мы можем привести создание ролевой модели, осуществление ее синтеза и разработку спецификации объектов».
2. «*Процесс разработки системы* охватывает обычный жизненный цикл программных продуктов, который начинается с формулирования потребностей пользователей, а заканчивается введением в действие и сопровождением системы, которая удовлетворяет эти потребности».
3. «*Процесс производства повторно используемых средств* — это наименее сформировавшийся процесс разработки программного обеспечения, однако мы надеемся, что в будущем именно от него будут в наибольшей степени зависеть эффективность и качество продуктов. Мы обращаем особое внимание на задачу непрерывного производства ряда близкородственных систем, строящихся из постоянно совершенствующегося набора компонентов многократного применения. Создание системы должно заключаться преимущественно в настройке и повторном использовании надежных, испытанных компонентов; для того чтобы доработать такую систему, ее, возможно, придется дополнить несколькими новыми компонентами».

Таким образом, одним из основополагающих принципов OOram является признание необходимости выделения процесса разработки для повторного использования.

Кроме того, Реенскауг и его соавторы делают крайне важное замечание о том, что создание повторно используемых объектов по многим параметрам напоминает процесс разработки изделия, чей жизненный цикл можно разделить на пять этапов [Ree96].

1. *Анализ рынка.* «Разработчик должен знать потребности потенциальных пользователей и сопоставлять их с затратами на альтернативные решения. Кроме того, для того чтобы повторно используемый компонент можно было применять на практике, разработчик должен понимать, в каких условиях работают потенциальные пользователи».
2. *Разработка продукта.* «Повторно используемый компонент нужно спроектировать, реализовать и протестировать в условиях одного или нескольких опытных применений».
3. *Упаковка продукта.* «Для упакованного компонента многократного применения крайне важно наличие документации. В документации должны содержаться описания технологических процессов применения компонента, процедур установки и другие технические данные».
4. *Маркетинг.* «Необходимо проинформировать пользователей нашего компонента многократного применения о его существовании и убедить их в том, что он им нужен».
5. *Применение.* «Повторно используемый компонент должен найти практическое применение, помочь его пользователям повысить качество продукции, снизить временные издержки и сократить расходование денежных средств».

В одном из четырех конкретных примеров, приведенных в издании [Ree96], описывается процесс разработки объектно-ориентированного каркаса. Его основные этапы таковы.

- Этап 1. Выявление круга потребителей и их потребностей.
- Этап 2. Проведение анализа затрат и выгод.
- Этап 3. Проведение обратной разработки существующих программ.
- Этап 4. Определение нового каркаса.
- Этап 5. Составление документации каркаса в виде моделей, описывающих способы разрешения проблем.
- Этап 6. Описание проектирования и реализации каркаса.
- Этап 7. Информирование группы потребителей.

Этапы 1–3 напоминают анализ предметной области и предполагают определение ее рамок, анализ заинтересованных лиц и существующих приложений. К сожалению, моделирование характеристик и характеристические модели в OOram не предусмотрены. Важность этих двух элементов мы будем обсуждать в главе 4. Этап 5 предполагает применение стандартной методики документирования каркасов при помощи моделей (см. также [Joh92, MCK97]).

3.6.3. Инициатива по разработке программного обеспечения на основе повторного использования (RSEB)

RSEB [JGJ97] — это объектно-ориентированный метод разработки программных продуктов, направленный на повторное использование; он основывается на нотации UML, методе объектно-ориентированной разработки программного обеспечения, описанном Джекобсоном (Jacobson) и его соавторами в работе [JCJO92], а также методе объектно-ориентированной реконструкции бизнес-процессов (OO Business Process Reengineering) [JEJ94]. Разработка данного метода проходила в компаниях Hewlett-Packard (М. Грисс (M. Griss)) и Rational Software (А. Джекобсон и П. Джонсон (P. Jonsson) — ранее они работали в Objectory AB). Замысел заключался в том, чтобы облегчить разработку и использование объектно-ориентированного программного обеспечения многократного применения. Подобно Унифицированному процессу, метод RSEB итеративен и ориентирован на элементы Use Case.

В первую очередь мы рассмотрим терминологию, принятую в RSEB [JGJ97].

- *Прикладная система.* Термин «прикладная система» в RSEB по смыслу соответствует понятию «программная система», которым мы пользуемся в настоящей книге. Разработчики RSEB отмечают, что «причина, по которой мы употребляем термин «прикладная система» вместо более широкого термина «приложение», кроется в том, что мы хотим подчеркнуть, что прикладные системы являются программными системами и определяются системными моделями».
- *Компонент.* «Компонент — это тип, класс или любое другое рабочее средство (например, элемент Use Case, модельный элемент анализа, проектирования

или реализации), разработка которого производится в расчете на многократное применение». Итак, понятие компонента, принятое в RSEB, соответствует нашему термину «повторно используемое средство».

- *Компонентная система.* Компонентная система — это системное изделие, содержащее ряд повторно используемых характеристик. По сравнению с прикладными системами компонентные системы носят более родовой характер, они лучше приспособлены к многократному применению и специализации; с другой стороны, их разработка требует больших усилий. В качестве примеров компонентных систем можно привести повторно используемые каркасы графических пользовательских интерфейсов и математические библиотеки; кроме того, существуют и более сложные компонентные системы, на основе которых можно производить целые прикладные системы. Если мы проанализируем класс прикладных систем и разобьем их на родовые подсистемы, окажется, что компонентные системы обеспечивают для них многократно используемые решения.

В RSEB предусмотрены отдельные процессы разработки для повторного использования (инженерии предметной области) и разработки с повторным использованием (то есть прикладной инженерии). Инженерия предметной области, по версии RSEB, состоит из двух процессов [JGJ97].

- *Конструирование семейства приложений.* Так называется процесс разработки и сопровождения общей многоуровневой архитектуры систем.
- *Конструирование компонентных систем.* Процесс разработки компонентных систем для различных частей прикладной системы, направленный на конструирование и упаковку надежных, расширяемых и гибких компонентов.

Прикладная инженерия (то есть процесс создания конкретных систем из повторно используемых средств), согласно терминологии RSEB, называется конструированием прикладных систем.

Разбиение инженерии предметной области на конструирование семейства приложений и конструирование компонентных систем обусловлено четким разделением задач по разработке общей архитектуры и многократно используемых решений для подсистем. В издании, посвященном RSEB, описывается типовой процесс конструирования компонентных систем на базе основных компонентов процесса объектно-ориентированного метода системной разработки. Это хорошее начало. Впрочем, в разделе 3.5 мы уже отмечали, что специализированные горизонтальные методы инженерии предметной области здесь несколько другие — это связано с тем, что не для всех подсистем подходят одни и те же стили и методики моделирования. Таким образом, в процессе конструирования семейства приложений различные специализированные горизонтальные методы инженерии предметной области, такие как DEMRAL (см. раздел 3.6.5), могут «вызываться» на правах методов конструирования компонентных систем в расчете на разные подсистемы (рис. 3.3).

В RSEB подробно разработан вопрос моделирования изменчивости. Понятие изменяемых параметров на уровне абстракций представлено как расширение нотации UML. Изменяемый параметр «обозначает одно или несколько мест, в которых будет наблюдаться изменчивость» [JGJ97]. Изменяемый параметр, подобно

элементу Use Case или компоненту, обозначается жирной точкой на элементе моделирования (рис. 3.4). К примеру, у компонента `Счет` есть изменяемый параметр `{Допущен овердрафт}`; с ним посредством простых ассоциаций UML связаны два вариантных компонента: `Удержание штрафа` и `Овердрафт не предусмотрен`. У элемента Use Case `Снятие денег со счета` также есть изменяемый параметр `{Допущен овердрафт}`. Два вариантных элемента Use Case — `Удержание штрафа` и `Овердрафт не предусмотрен` — связаны с ним через отношение `<<extends>>`. Каждый из этих вариантных элементов описывает, что должно случиться, если будет допущен овердрафт. В разделе 4.4.1.7 мы расширим понятие изменяемых параметров и поговорим о различных категориях изменчивости.

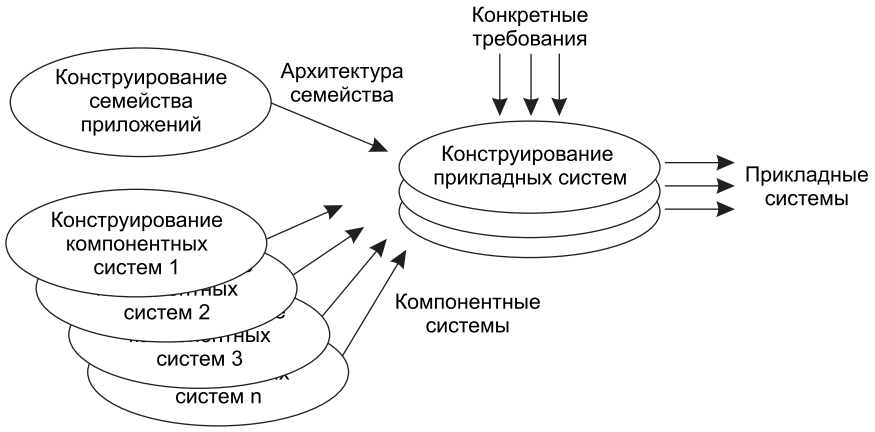

Рис. 3.3. Поток артефактов в RSEB при участии специализированных процессов конструирования компонентных систем

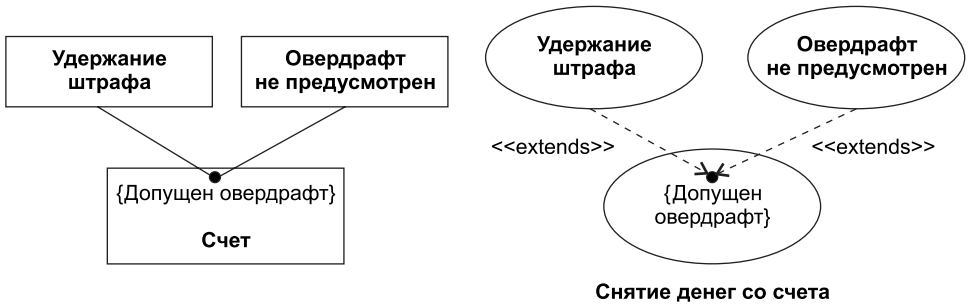

Рис. 3.4. Изменяемые параметры компонентов и элементов Use Case (видоизмененный вариант схемы из издания [JGJ97])

При помощи механизмов изменчивости осуществляется реализация изменяемых параметров в рамках более конкретных моделей. Любая модель, задействованная в технологическом процессе, должна обеспечивать поддержку того или иного механизма изменчивости. Примеры механизмов изменчивости приводятся в табл. 3.1.

RSEB обладает такой существенной возможностью, как моделирование изменчивости в элементах Use Case. В издании, описывающем RSEB, указывается на несколько важных аргументов в пользу обеспечения изменчивости в рамках моделей Use Case.

- Корректировка пользовательских или системных интерфейсов.
- Ссылки устанавливаются на разные типы объектов — к примеру, денежные средства могут сниматься со счета до востребования или с общего счета.
- Возможность выбора различных функциональных возможностей и наличие дополнительных функций.
- Корректировка ограничений и бизнес-правил.
- Обнаружение ошибок.
- Различия по части производительности и масштабируемости.

В ходе моделирования вариантов использования можно задействовать такие механизмы изменчивости, как шаблоны элементов Use Case, макрокоманды и параметры, делегирование элементов Use Case (то есть отношения использования) и расширения элементов Use Case [JGJ97].

Основной принцип RSEB заключается в том, чтобы поддерживать явно отслеживаемые связи представления изменчивости во всех моделях — таким образом, изменчивость в элементах Use Case можно проследить до изменчивости в рамках объектных моделей анализа, проектирования и реализации. В UML отслеживаемые связи моделируются при помощи отношения зависимости `<<trace>>`.

Поскольку, как мы уже отмечали, RSEB основывается на объектно-ориентированном методе системной разработки (OOSE), каждый из трех его процессов — конструирование семейства приложений, конструирование компонентных систем и конструирование прикладных систем — распадается на пять основных компонентов процесса OOSE.

- Фиксация требований.
- Анализ робастности.
- Проектирование.
- Реализация.
- Тестирование.

Таблица 3.1. Некоторые механизмы изменчивости (видоизмененный вариант схемы, приведенной в издании [JGJ97])

Механизм	Тип изменяемого параметра	Тип варианта	Частные случаи применения
Наследование	Виртуальная операция (virtual operation)	Подкласс или подтип	Специализация и добавление отдельных операций при сохранении всех остальных
Расширения	Точка расширения (extension point)	Расширение	Одновременное подключение нескольких вариантов в каждом из изменяемых параметров

продолжение ⇩

Таблица 3.1 (продолжение)

Механизм	Тип изменяемого параметра	Тип варианта	Частные случаи применения
Использование	Точка использования	Элемент Use Case	Повторное использование абстрактного элемента Use Case с целью создания специализированного элемента Use Case
Конфигурация	Гнездо элемента конфигурации	Элемент конфигурации	Отбор альтернативных функций и реализаций
Параметры	Параметр	Связанный параметр	Выбор в пользу тех или иных альтернативных характеристик
Конкретизация шаблона	Параметр шаблона	Экземпляр шаблона	Адаптация типов или выбор альтернативных блоков кода
Генерация	Параметр или языковой сценарий	Связанный параметр или выражение	Крупномасштабная разработка одного или нескольких типов или классов на основе проблемно-ориентированного языка

У процессов конструирования компонентных и прикладных систем есть и шестой компонент — упаковка; ее назначение заключается в упаковке компонентной системы (документировании, архивации с расчетом на последующий поиск) или прикладной системы (документировании, разработке сценариев инсталляции и т. д.).

К сожалению, несмотря на то что в рамках метода RSEB изменчивости уделено довольно большое внимание, компоненты процессов конструирования семейства приложений и компонентных систем не предполагают проведения таких аналитических действий, как определение предметной области и моделирование характеристик.

Определение предметной области в рамках процесса конструирования семейства приложений могло бы выполнять функцию ограничения рамок семейства приложений или линейки продуктов с учетом предпочтений имеющихся и потенциальных заинтересованных лиц. Эту задачу называют *определением области действия семейства приложений* (application family scoping). Мы уже обсуждали вопросы, связанные с определением области действия семейств приложений, применительно к линейкам и семействам продуктов (см. раздел 2.7.2).

Аналогичные действия, которые следует проводить в рамках конструирования компонентных систем, обозначаются как *компонентно-ориентированное определение области действия* (component-oriented scoping). С этим процессом, в частности, связаны следующие проблемы.

- Возможность повторного использования компонентной системы в различных линейках или семействах продуктов одной организации.
- Возможность продажи этой компонентной системы клиентам вне организации.
- Технические вопросы — к примеру, характер данной предметной области и необходимые методики моделирования.
- Организационные вопросы наподобие комплектования штатов и параллельного конструирования.

Компонентно-ориентированное определение предметной области аналогично декомпозиции участников семейства приложений на родовые подсистемы; этим объясняется важность выполнения этого процесса в ходе разработки архитектуры семейства приложений.

Еще одним недостатком метода RSEB представляется отсутствие в нем характеристических моделей. Изменчивость в рамках RSEB выражается на высшем уровне в виде изменяемых параметров (это особенно заметно по элементам Use Case), которые впоследствии при помощи различных механизмов изменчивости реализуются в других моделях. В работе [GFA98] Грисс и его соавторы высказывают мнение о том, что этих методик моделирования на практике оказывается недостаточно. Свой опыт применения метода, основанного исключительно на элементах Use Case, в рамках предметной области телекоммуникаций они описывают следующим образом.

> Отсутствие явного представления характеристик при работе в отрасли телекоммуникаций создает исключительно серьезные проблемы; не помогает даже возможность моделирования вариантов использования. Во-первых, модели Use Case не способны детально воспроизводить многие характеристики реализации и технические характеристики, которые в системах телефонной связи являются общеупотребительными (пример — «типы коммутаторов»). Во-вторых, для описания телекоммуникационных услуг часто требуется огромное количество элементов Use Case; в том же случае, когда элементы Use Case параметризируются при помощи изменяемых параметров RSEB, содержащих описания множества расширений, вариантов и параметров, разработчику предметной области в процессе создания новых систем ничего не стоит окончательно запутаться. Последующие пользователи, скорее всего, будут сбиты с толку вопросом о том, для каких прикладных систем предназначены те или иные характеристики и элементы Use Case.

Учитывая этот печальный опыт, они разработали метод FeatuRSEB [GFA98]; для того чтобы снять все перечисленные проблемы, они расширили RSEB за счет точного определения предметных областей, операций моделирования характеристик и характеристических моделей. Этот новый метод рассматривается в следующем разделе.

3.6.4. FeatuRSEB

Силами компаний Hewlett-Packard и Intecs Sistemi метод FODAcom [VAM+98] — объектно-ориентированный вариант FODA для предметной области телекоммуникаций — был интегрирован в RSEB; в результате получился новый метод под названием FeatuRSEB [GFA98].

Преимущества FeatuRSEB в сравнении с RSEB находятся в двух существенных областях.

- Процессы конструирования семейств приложений и компонентных систем снабжены возможностями точного определения предметной области и ее планирования, а также операциями моделирования характеристик.
- В качестве основного средства представления сходств, изменчивостей и зависимостей выступают характеристические модели.

Унифицированный процесс и RSEB основываются на модели представлений архитектуры «4+1», разработанной Крюхтеном (см. рис. 3.1); она предполагает

разработку нескольких моделей и одной дополнительной модели, выполняющей функцию объединения всех остальных. В рамках унифицированного процесса и RSEB в качестве такой объединяющей модели «+1» выступала модель Use Case; в FeatuRSEB это характеристическая модель. Грисс и его соавторы отмечают, что характеристическая модель «в сжатой форме обобщает изменчивость и общность, которые представлены в остальных моделях RSEB — главным образом, в модели Use Case» [GFA98]. Другими словами, метод FeatuRSEB *ориентирован на характеристическую модель* (feature model centric). Затем они декларируют принцип FeatuRSEB: «не все то, что *может* быть характеристикой, *должно* ею быть. Описания характеристик должны быть устойчивыми и ясными. Характеристики используются главным образом для того, чтобы мы могли проводить различия между отдельными *вариантами выбора* (choices); задачу подробного описания функциональных возможностей выполняют модель Use Case и объектная модель».

Помимо этого, Грисс и соавторы рассказывают о существенной связи между моделями Use Case и характеристическими моделями. По их мнению, в отличие от модели Use Case, которая фиксирует требования к системе с точки зрения пользователя (так называемые «рабочие требования»), модель характеристическая упорядочивает требования с позиции последующих пользователей, основываясь при этом на результатах анализа изменчивости и общности (а также анализе зависимостей, добавим мы от себя).

Как и их аналоги в FODA, характеристические модели FeatuRSEB состоят из диаграмм характеристик, снабженных сведениями об ограничениях, времени связывания, категории, логическом обосновании и т. д.

Моделирование характеристик в FeatuRSEB состоит из следующих этапов [GFA98].

1. Объединение отдельных экземпляров модели Use Case в доменную модель Use Case (в RSEB она называется моделью Use Case семейства). Фиксация и изображение различий при помощи изменяемых параметров. Отслеживание порождающих экземпляров посредством `<<trace>>`.
2. Создание начальной характеристической модели при помощи *функциональных характеристик* (functional features), выведенных из доменной модели Use Case (обычно в качестве основы для присвоения имен характеристикам используются имена элементов Use Case).
3. Создание объектной модели анализа RSEB; расширение характеристической модели за счет *архитектурных характеристик* (architectural features). Эти характеристики больше связаны не с конкретной функцией, а со структурой и конфигурацией системы.
4. Создание проектной модели RSEB; расширение характеристической модели за счет *характеристик реализации* (implementation features).

Итак, в рамках метода FeatuRSEB существуют функциональные характеристики, архитектурные характеристики и характеристики реализации.

Кроме того, Грисс и его соавторы предлагают вариант реализации нотации диаграмм характеристик на UML и приводят некоторые требования относительно инструментальной базы для характеристических моделей. Эти две темы мы рассмотрим в разделе 4.7.

3.6.5. Метод проектирования алгоритмических библиотек многократного применения на основе инженерии предметной области (DEMRAL)

DEMRAL — это метод инженерии предметной области, предназначенный для разработки алгоритмических библиотек — в частности, библиотек численного анализа, библиотек контейнеров, библиотек обработки и распознавания изображений, библиотек распознавания речи, библиотек графовых вычислений и т. д. Очевидно, что это горизонтальный метод. Основными видами абстракций в алгоритмических предметных областях являются абстрактные типы данных (abstract data types, ADT) и алгоритмы. Основными проектными задачами DEMRAL являются обеспечение высокой производительности, эффективное представление бессчетных вариантов ADT и алгоритмов, достижение наилучшей адаптируемости и возможности повторного использования, а также предоставление абстрактного интерфейса. DEMRAL задумывался как специализация метода ODM (см. раздел 2.8.2).

Основным свойством и движущей силой DEMRAL является моделирование характеристик. DEMRAL предполагает создание высокоуровневой характеристической модели данной предметной области, а также характеристических моделей каждого из относящихся к ней понятий. Некоторые понятия DEMRAL заимствованы из аспектно-ориентированного программирования — в частности, аспектная декомпозиция и изображение различных аспектов средствами предметно-ориентированных языков (domain-specific languages, DSLs) (см. главу 8).

Одним из существенных понятий DEMRAL являются *предметно-ориентированные языки конфигурирования* (configuration DSLs). При помощи таких языков осуществляется конфигурирование компонентов. Они допускают различные уровни управления, в результате чего клиенты получают возможность формулировать свои потребности с оптимальной степенью детализации. В DEMRAL предусмотрен метод выведения предметно-ориентированных языков конфигурирования из характеристических моделей.

Кроме того, DEMRAL дает советы по поводу реализации доменных моделей посредством объектно-ориентированных методик и метапрограммирования (к примеру, генераторов, трансформационных систем или встроенных в языки программирования возможностей метапрограммирования).

Подробное описание DEMRAL приводится в главе 5.