С. С. Девятов

Проектирование программного обеспечения с использованием стандартов UML 2.0 и SysML 1.0

Инженерия программного обеспечения для достижения положительных результатов требует применения специализированных методик управления, как самим процессом, так и всеми стадиями анализа, проектирования, реализации и последующего использования разработанного продукта. В статье рассматривается основа данного процесса — унифицированный язык моделирования UML. В качестве основных способов его применения выбрано управление процессами разработки приложений в целом, а также рассмотрены вопросы моделирования систем на основе языка SysML.

Разработка программного обеспечения: задачи и решения

Разработка программного обеспечения (ПО) является трудно формализуемым и одновременно творческим процессом. Разработчики вынуждены решать две основные задачи: построение программного комплекса в определенные сроки в соответствии с требованиями заказчика и поиск оптимальной реализации компонентов приложения.

В связи с тем что отрасль инженерии ПО является одной из самых динамично развивающихся в сфере IT и демонстрирует существенный ежегодный прирост объемов рынка реализации собственной продукции, важно рассмотреть технологии, превращающие «процесс творческого поиска» в «инженерную науку» со всеми присущими ей атрибутами.

При разработке ПО мы будем ориентироваться на основные составляющие, которые приведены в работе Э. Брауде [1]:

- продукт,
- персонал,
- процесс,
- проект,
- артефакты.

Кратко определим приведенные категории.

Продукт — то, что должно быть получено в результате деятельности разработчиков ПО. К продукту изначально предъявляются определенные требования, которые лимитируют «полет фантазии» разработчиков. Продукт состоит из программного кода, документации и различных артефактов.

Персонал — в первую очередь, это коллектив разработчиков различной специализации. Кроме того, к персоналу относят заказчиков, пользователей, инвесторов.

Процесс — последовательность этапов анализа, проектирования, разработки, внедрения и поддержки в ходе создания продукта.

Проект — совокупность действий, необходимая для создания продукта.

Артефакты — дополнительные результаты, полученные в ходе разработки ПО, ожидаемые или полученные случайным образом.

Обратим внимание на существенные аспекты проектирования.

Наиболее распространенной парадигмой анализа, проектирования и программирования на текущий момент является объектно-ориентированная парадигма. Большинство современных RAD-сред (Rapid Application Development) построены с использованием именно такой парадигмы.

При разработке ПО (равно как и при реализации прочих проектов) требуется применение стандартов, позволяющих реализовать не просто один отдельно взятый проект, а создать конвейер по разработке. Как пра-

вило, под подобными стандартами понимают технологии, принятые индустрией и рынком — промышленные стандарты. В качестве стандарта на инструментарий описания моделей при разработке ПО на текущий момент широко используется унифицированный язык моделирования UML (Unified Modeling Language). Особо следует отметить два факта:

- UML не методология, а нотация описания моделей;
- UML является расширяемой концепцией описания моделей не только в области программирования, но и в области системного анализа, моделирования технических и экономических систем, а также их динамики (стандарт-диалект SysML (System Modeling Language)).

В данной статье за основу описания методов использования UML при разработке ПО выбран коллективный процесс разработки программного обеспечения TSP (Team Software Process). Необходимость в использовании TSP-методологии вместо индивидуального процесса разработки PSP (Personal Software Process) обусловлена невозможностью работы одного программиста при разработке современного ПО. Рассмотрим основные проблемы, которые должны быть решены с помощью TSP:

- сбор команды разработчиков — группа должна быть самоуправляемой, для этого необходимо определить собственные цели, составить процесс и планы достижения поставленных целей, а также отслеживать ход выполнения отдельных этапов работ;
- дать ответ руководителю проекта о способе управления командами: обеспечить согласование деятельности команд, предоставить участникам персональные инструкции, мотивировать к скорейшему достижению результатов и обеспечить максимальную производительность до завершения проекта;
- обеспечить уверенное достижение уровней модели зрелости возможностей CMM (Capability Maturity Model);
- определить направление и способы улучшения процесса разработки для успешных команд;
- содействовать подготовке квалифицированных специалистов для работы в IT-отрасли.

В качестве основных описаний процесса разработки используется модель унифицированного процесса разработки программного обеспечения USDP (Unified Software Development Process). Данный процесс (был разработан и описан Г. Бучем, А. Якобсоном и Дж. Рамбо) имеет следующие стадии:

- предварительный анализ требований,
- детальный анализ требований,
- проектирование,
- реализация,
- тестирование.

С точки зрения фактора времени, процесс проходит следующие фазы:

- начало,
- проектирование,
- конструирование,
- переход.

Фазы проектирования, конструирования и перехода — многоитеративны. В соответствии с USDP выделяются следующие модели: использования, аналитическая, проектирования, развертывания, реализации и тестирования.

Опишем последовательность этапов разработки программного проекта.

1. Определение концепции системы (концептуализация системы). Определяется предметная область и предназначение для разрабатываемого приложения, формулируются предварительные требования.

2. Анализ предметной области. Детализированное рассмотрение требований производится путем построения модели (или системы моделей). Цель анализа — определе-

ние того, что должно быть сделано. Необходимо получить полное представление о задаче, прежде чем приступать к ее решению.

3. Проектирование системы. Формулируется и документируется архитектура системы, определяющая основы для последующего проектирования классов.

4. Проектирование классов. Модели реальной системы, полученные на этапе анализа, расширяются и корректируются таким образом, чтобы они могли быть реализованы компьютерной программой. Определяются алгоритмы для реализации отдельных операций и процессов.

5. Реализация. Проекты классов системы преобразуются в программный код и/или структуры баз данных.

6. Тестирование. Выполняется проверка на пригодность для практического использования и удовлетворения поставленных изначально требований.

7. Обучение персонала. Производится обучение персонала для работы с компонентами системы.

8. Развертывание. Компоненты системы устанавливаются на компьютерах пользователей.

9. Поддержка. Реализуется комплекс мероприятий по обеспечению стабильного функционирования ПО: поддержка пользователей, поиск вновь обнаруженных ошибок, реинжиниринг ПО и бизнес-процессов.

Процесс разработки — непрерывен. Модели постоянно уточняются и оптимизируются, даже в случаях перехода от анализа к проектированию, а затем к реализации. На всех этапах разработки ПО используют одни и те же выбранные изначально концепции проектирования и системы обозначений. Разница в том, что вначале процесса внимание в основном уделяется потребностям заказчика, а в конце — вычислительным ресурсам.

Объектно-ориентированный подход к построению ПО переносит основные усилия по разработке программного обеспечения на этапы анализа и проектирования. В ряде ситуаций может возникнуть мнение о нецелесообразности затрат существенного объема времени на анализ и проектирование системы. Однако эти затраты окупаются быстрой и простой реализацией, так как построенное в соответствии с объектно-ориентированной концепцией программирования программное обеспечение оказывается более понятным и легко адаптируемым, проще осуществлять его поддержку и вносить изменения в соответствии с изменениями реального мира.

В данной статье рассматриваются вопросы анализа, проектирования и разработки. Вопросы тестирования, обучения, развертывания и поддержки детально не рассматриваются, так как они выходят за рамки непосредственно проектных работ, а относятся, скорее, к вопросам проектирования информационных систем. Вопросы тестирования составляют набор специализированных технологий и методик, что также не позволяет их рассмотреть в рамках одного материала.

Понятийный аппарат UML

Стереотип — новый вид элемента модели, который наследует структуру существующего элемента, но имеет ряд дополнительных ограничений, обладает другой интерпретацией и обозначением. Стереотип содержит набор теговых величин.

Теговая величина — это пользовательский атрибут, который определяется для элементов модели, а не для объектов в работающей системе. Она, например, может нести информацию об управлении проектом, указания по генерации кода и прочие сведения.

Ограничение — формализованное условие, выраженное текстовой строкой, с помощью некоторого языка ограничений. В UML используется язык OCL (Object Constrained Language).

Профиль — набор стереотипов и ограничений с определенным предназначением, который применяется в пользовательских пакетах.

История возникновения UML, обзор технологии

UML представляет собой язык визуального моделирования для решения различных задач, который может применяться при определении, визуализации, конструировании и документировании артефактов, моделей или программной среды. UML не определяет конкретный процесс разработки или концепцию анализа и проектирования. Изначально, UML был создан на основе метода Fusion, который в свою очередь объединил концепции OMT (Object Modeling Technique) и CRC (Class Responsibility Collaborator). Основоположники данного метода — Г. Буч («нотация Буча»), Дж. Рамбо и позднее присоединившийся к ним А. Якобсон. Все они работали в компании *Rational Software*. Потом, в результате объединения усилий с командой OMG (Object Management Group), была разработана конечная версия UML, которая получила статус версии UML1. UML2 основывался на предыдущем опыте разработки унифицированного языка и позволил применять технологию для использования не только в области программной инженерии, но и в других предметных областях (моделирование экономических объектов, организационных структур и т.д.). Основные отличия версий UML1 и UML2 заключаются в:

- применении новых стандартов при конструировании диаграмм последовательности, что было сделано для придания им объектно-ориентированного вида;
- конечные автоматы не используются для моделирования деятельности, а применяется общепризнанная в моделировании бизнес-процессов нотация;
- моделирование деятельности унифицировано с моделированием действий, что формирует полностью процедурную модель;
- введены конструкции контекстного моделирования для классов и кооперации, что позволяет использовать свободную или строгую инкапсуляцию, а также формирование внутренних структур из более мелких элементов;
- компоненты рассматриваются как конструкции проектирования, а артефакты — как физические элементы, подлежащие развертыванию.

Внутренние механизмы UML, образующие метамодель языка, формируют следующую картину:

- структура ядра UML унифицирована с компонентами MOF (Meta-Object Facility), которая в свою очередь относится к концептуальному моделированию;
- метамодель реструктуризирована для устранения избыточных конструкций и обеспечения возможности создания новых подмножеств UML в других областях (например, SysML);
- введена технология профилей, позволяющая определять специфичные для конкретных предметных областей и технологий расширения.

Под унификацией в языке UML подразумевают унификацию:

- старых методов и нотаций,
- по этапам разработки,
- по предметным областям,
- по платформам и языкам реализации,
- по процессам разработки,
- внутренних концепций.

В целом, все концепции и модели UML можно отнести к четырем областям: статическая структура, элементы проектирования, элементы развертывания, динамическое поведение.

Статическая структура позволяет определить полное множество объектов, их внутренние свойства и отношения между собой. Концепции приложения моделируются как классы, описывающие некоторый тип объектов, которые содержат определенную информацию и взаимодействуют для реализации некоторого поведения.

Элементы проектирования представляют собой некоторые конструкции модели, предназначенные как для логического ана-

лиза, так и для проектирования, обеспечивающего реализацию систем. Структурированный классификатор содержит реализацию класса в виде набора элементов, связанных между собой соединителями. Кооперация позволяет моделировать набор объектов. Компонент — замещаемая часть системы, соответствующая некоторому набору интерфейсов и предназначенная для их реализации.

Элементы развертывания состоят из узлов, артефактов и представления развертывания системы. Узел — вычислительный ресурс времени исполнения, определяющий местонахождение исполняемых компонентов и объектов. Артефакт — физическая единица информации или описание поведения вычислительной системы. Артефакты развертываются в узлах. Представление развертывания системы позволяет описать конфигурацию узлов системы и расположение артефактов в этих узлах.

Динамическое поведение реализуется с помощью одного из трех способов его моделирования (поведения):

- описание истории жизни объекта и его взаимодействия с внешней средой,
- шаблоны взаимодействия для набора связанных объектов при реализации определенной модели поведения,
- описание эволюции процесса выполнения программы в ходе ее эксплуатации.

После определения основных концепций UML становится возможным оценить влияние методологии на управление процессом разработки ПО и моделирование программных компонент.

Рассмотрим этапы процесса разработки более подробно.

Концептуализация системы (system conception) означает возникновение «идеи» приложения. В самом начале существования концепции кто-то формирует идеологию, составляет проект реализации и предлагает ее потенциальному заказчику и пользователю. На данном этапе разработчик должен точно представлять потребности возможных потребителей продукта. Вместе с тем должна присутствовать точная оценка технологических возможностей разработчиков.

Анализ (analysis) — по своей сути, объединяет процессы исследования и моделирования. Задача аналитиков — исследование и формализация требований. Выполняется описание продукта, а не процесса или способа его реализации. Анализ является ключевой и наиболее сложной задачей. Разработчики должны полностью осознать сложность ставящейся перед ними задачи перед тем как заняться разрешением дополнительных вопросов, которые неизбежно возникнут на этапе реализации. Надежные модели — обязательное условие для создания расширяемого, эффективного, надежного и корректного приложения. Никакие исправления на этапах реализации не смогут исправить внутреннюю несогласованность приложения и скомпенсировать недостаток предусмотрительности.

В процессе анализа разработчики используют все доступные источники информации (документы, интервью, аналогичные и связанные приложения) и устраняют возникающие неопределенности. Часто бизнес-эксперты оказываются не в состоянии сформулировать точные требования, и их уточнением приходится заниматься разработчикам. Моделирование ускоряет сходимость представлений разработчиков и бизнес-экспертов, потому что гораздо быстрее работать с различными вариантами моделей, чем с различными реализациями кода. Модели подчеркивают недостатки и несогласованные моменты, которые можно затем исправить. По мере того как разработчики прорабатывают и уточняют модель, она становится все более согласованной.

Этап анализа делится на две стадии. Сначала осуществляется анализ предметной области, а затем — анализ приложения. Анализ предметной области применяется к объектам реального мира, семантику которых охватывает приложение. Анализ предметной области выделяет понятия и отношения, а функциональность при этом неявным образом содержится в модели классов.

Конструирование модели предметной области сводится, главным образом, к принятию решений о том, какую информацию следует отразить в модели и как ее нужно представить.

За анализом предметной области следует анализ приложения, который относится к компьютерным аспектам приложения, видимым пользователям. Модель приложения не предписывает конкретной реализации системы. Она описывает внешний вид приложения, которое воспринимается как черный ящик. Классы приложения нельзя определить на этапе анализа, но их часто можно взять из предыдущих приложений.

В процессе *проектирования системы* (system design) разработчик принимает стратегические решения с наиболее широким спектром последствий. Он должен сформулировать архитектуру системы и выбрать глобальные стратегии и политики, определяющие последующие, более детализированные этапы проектирования.

Архитектура — это высокоуровневый план, или стратегия решения задачи по созданию приложения. Ее выбор определяется требованиями и предшествующим опытом разработчика. По возможности, архитектура должна включать исполняемый скелет, который можно было бы тестировать. Проектировщик должен представлять, каким образом новая система будет взаимодействовать с уже существующими.

На этапе *проектирования классов* (class design) разработчик расширяет и оптимизирует аналитические модели. Происходит смещение точки приложения усилий с концепций приложения на компьютерные концепции. Разработчики выбирают алгоритмы реализации основных функций системы, однако они должны воздерживаться от выбора конкретных языков программирования.

Реализация (implementation) — это этап написания реального кода. Разработчики реализуют элементы проекта на языках программирования, создавая программный код и структуру базы данных. Достаточно часто код может частично генерироваться автоматически из проектной модели.

После реализации система завершена, но до эксплуатации она должна пройти обязательный этап *тестирования.* Тестирование позволяет также раскрыть случайные ошибки, которые появились в системе в процессе ее создания. Если приложение должно работать на разном оборудовании или под разными операционными системами (на разных платформах), оно должно быть проверено на всех этих платформах.

Организация должна обучать пользователей, чтобы они могли полностью использовать все преимущества нового приложения. Обучение ускоряет усвоение пользователями новых навыков. Отдельная команда должна готовить пользовательскую документацию параллельно с разработкой программного обеспечения. Отдел контроля качества может сравнивать программное обеспечение с пользовательской документацией. Это позволяет гарантировать, что программа соответствует исходным требованиям.

При развертывании системы (deployment) на предприятии пользователя можно ожидать различных эффектов взаимодействия с другими системами, платформами и конфигурациями. Разработчики должны оптимизировать систему под разные нагрузки, оформить сценарии и процедуры установки. Некоторые клиенты потребуют настройки системы под себя. Персонал должен также выполнить локализацию продукта на разные языки с учетом пользовательской *локали* (locale — переменная, определяющая пользовательские данные локализации, т. е. специфические обозначения единиц измерения, форматы дат и времени, и т. д.). В результате происходит *выпуск продукта* (release).

После завершения разработки и развертывания системы команда переходит к этапу *поддержки.* Поддержка подразумевает несколько видов деятельности. В процессе эксплуатации выявляются ошибки, содержавшиеся в программном обеспечении и не выявленные на этапе тестирования. Эти ошибки необходимо исправлять. Успешное приложение вызывает запросы

на усовершенствование, а долгоживущее приложение — на реструктуризацию.

Управление проектами

Управление проектами — особая область менеджмента, применение которой дает ощутимые результаты. Профессионалов в этой области высоко ценят, а сама методология управления проектами стала фактическим стандартом управления на многих предприятиях и применяется в той или иной степени практически во всех крупных организациях.

Управление проектами дает ощутимые результаты во всех областях приложений, чем и объясняется растущая популярность этой технологии. Для руководителей информационных служб она представляет интерес и как технология, которую полезно внедрить на своих предприятиях, и как средство управления собственными проектами, к которым можно отнести и разработку программного обеспечения, и внедрение тех или иных информационных систем, а также прочие изменения, носящие уникальный характер и вре́менные по своей природе.

Моделирование ПО

Создание документации

При разработке ПО создается большой объем разнообразной документации. Она необходима для передачи информации между разработчиками ПО, как средство управления разработкой ПО и передачи пользователям информации, необходимой для применения и сопровождения ПО. На создание этой документации приходится большая доля стоимости ПО.

Эту документацию можно разбить на две группы:

- документы управления разработкой ПО;
- документы, входящие в состав ПО.

Документы управления разработкой ПО (process documentation), протоколируют процессы разработки и сопровождения ПО, обеспечивая связи внутри коллектива разработчиков и между коллективом разработчиков и *менеджерами* (managers), т. е. лицами, управляющими разработкой. Эти документы могут быть следующих типов:

- *планы, оценки, расписания*. Они создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения;
- *отчеты об использовании ресурсов в процессе разработки*. Создаются менеджерами;
- *стандарты*. Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПО. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка данного ПО;
- *рабочие документы*. Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПО;
- *заметки и переписка*. Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

Документы, входящие в состав ПО (product documentation), описывают ПО как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПО). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПО (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами). Во всяком случае они должны быть проверены (протестированы) на соответствие программам ПО. Эти документы образуют два комплекта с разным назначением:

- пользовательская документация ПО,
- документация по сопровождению ПО.

Внутренняя документация разработчика

Документация по сопровождению ПО (system documentation) описывает ПО с точки зрения ее разработки. Эта документация необходима, если ПО предполагает изучение того, как оно устроено (сконструировано), и модернизацию его программ.

Документация по сопровождению ПО можно разбить на две группы:

- документация, определяющая строение программ и структур данных ПО и технологию их разработки;
- документация, помогающая вносить изменения в ПО.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПО. Она включает следующие документы:

- внешнее описание ПО (requirements document);
- описание архитектуры ПО (description of the system architecture), включая внешнюю спецификацию каждой ее программы;
- для каждой программы ПО — описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля;
- для каждого модуля — его спецификация и описание его строения (design description);
- тексты модулей на выбранном языке программирования (program source code listings);
- документы установления достоверности ПО (validation documents), описывающие как устанавливалась достоверность каждой программы ПО и как информация об установлении достоверности связывалась с требованиями к ПО.

Документы установления достоверности ПО включают прежде всего документацию по тестированию (схему тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки ПО, например, доказательства свойств программ.

Документация второй группы содержит руководство по сопровождению ПО (system maintenance guide), которое описывает известные проблемы вместе с ПО, описывает, какие части системы являются аппаратно- и программно-зависимыми, и как развитие ПО учтено в расчете его строения (конструкции).

Общая проблема сопровождения ПО — обеспечить, чтобы все его представления оставались согласованными, когда ПО изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть зафиксированы в базе данных управления конфигурацией.

Документация пользователя

Пользовательская документация ПО (user documentation) объясняет пользователям, как они должны действовать, чтобы применить данное ПО. Она необходима, если ПО предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми руководствуется пользователь при *инсталляции* ПО (при установке ПО с соответствующей настройкой на среду применения ПО), при применении ПО для решения своих задач и управлении ПО (например, когда данное ПО взаимодействует с другими системами). Эти документы частично затрагивают вопросы сопровождения ПО, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей ПО: ординарных пользователей и администраторов ПО. *Ординарный пользователь ПО* (end-user) использует его для решения своих задач в нужной предметной области. Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПО. Он может и не знать многих деталей работы компьютера или принципов программирования. *Администратор ПО* (system administrator) управляет использованием ПО ординарными поль-

зователями и осуществляет его сопровождение, не связанное с модификацией программ. Например, он может регулировать права доступа к ПО между ординарными пользователями, поддерживать связь с его поставщиками или выполнять определенные действия, чтобы поддерживать ПО в рабочем состоянии, если оно является частью другой системы.

Состав пользовательской документации зависит от аудиторий пользователей, на которые ориентировано данное ПО, и от режима использования документов. Под *аудиторией* здесь понимается контингент пользователей, у которого есть необходимость в определенной пользовательской документации ПО. Удачный пользовательский документ существенно зависит от точного определения аудитории, для которой он предназначен. Пользовательская документация должна содержать информацию, необходимую для каждой аудитории. Под *режимом использования* документа понимается способ, определяющий каким образом применяется этот документ. Обычно пользователю достаточно больших программных систем требуются либо документы для изучения ПО (использование в виде *инструкции*), либо для уточнения некоторой информации (использование в виде *справочника*).

В соответствии с работами перечислим состав пользовательской документации для достаточно больших ПО.

- *Общее функциональное описание ПО*. Дает краткую характеристику функциональных возможностей ПО. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПО.
- *Руководство по инсталляции ПО*. Предназначено для системных администраторов. Оно должно детально предписывать, как устанавливать систему в конкретной среде. Включает описание машинно-считываемого носителя, на котором поставляется ПО, файлы, содержащие ПО, и требования к минимальной конфигурации аппаратуры.
- *Инструкция по применению ПО*. Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПО, описанную в форме удобной для ее изучения.
- *Справочник по применению ПО*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПО, описанную в форме удобной для избирательного поиска отдельных деталей.
- *Руководство по управлению ПО*. Предназначено для системных администраторов. Оно должно описывать сообщения, генерируемые когда ПО взаимодействует с другими системами, и как реагировать на эти сообщения. Кроме того, если ПО использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПО. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПО, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПО, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

«Обратный» инжиниринг (восстановление описания по исходному коду ПО)

«Обратный» инжиниринг — это процесс анализа программного обеспечения с целью идентификации программных компонент и связей между ними, а также формирования представления о нем, с дальнейшей перестройкой в новой форме (уже, в процессе

реинжиниринга). «Обратный» инжиниринг является *пассивным*, предполагая отсутствие деятельности по изменению или созданию нового программного обеспечения. Обычно, в результате усилий по «обратному» инжинирингу создаются модели вызовов (call graphs) и потоков управления (control flow graphs) на основе исходного кода системы. Один из типов «обратного» инжиниринга — создание новой документации на существующую систему (redocumentation). Другой из распространенных типов — восстановление дизайна системы (design recovery). *Рефакторинг* — трансформация программного обеспечения, в процессе которой программная система реорганизуется (не переписываясь) с целью улучшения структуры, без изменения поведения. Сохранение «формы» (платформы, архитектурных и технологических решений) существующей программной системы позволяет рассматривать рефакторинг как один из вариантов «обратного» инжиниринга.

Этапы «обратного» инжиниринга:

- *формирование представления об эксплуатируемой/сопровождаемой системе* включает восстановление, в первую очередь, бизнес- и функциональных требований к системе;
- *восстановление детального дизайна системы* включает идентификацию всех компонентов системы и создание модели вызовов и других связей между компонентами;
- *рефакторинг* как возможный процесс структурных изменений, вносимых в систему, в частности, для улучшения возможностей по ее дальнейшему сопровождению (включая модификацию, связанную с расширением функциональности);
- *переработка системы,* т.е. создание нового релиза/версии системы в той же форме (например, с использованием той же технологической платформы), что и текущая (эксплуатируемая) версия;
- *создание новой системы* рассматривает текущую версию и систему в целом как устаревшую (legacy).

Этапы проектирования ПО с применением UML

UML обеспечивает поддержку всех этапов жизненного цикла ПО и предоставляет для этих целей ряд графических средств.

На этапе создания концептуальной модели для описания бизнес-деятельности используются модели бизнес-прецедентов и диаграммы видов деятельности, для описания бизнес-объектов — модели бизнес-объектов и диаграммы последовательностей.

На этапе создания логической модели ПО описание требований к системе задается в виде модели и описания системных прецедентов, а предварительное проектирование осуществляется с использованием диаграмм классов, диаграмм последовательностей и диаграмм состояний.

На этапе создания физической модели детальное проектирование выполняется с использованием диаграмм классов, диаграмм компонентов и диаграмм развертывания.

Ниже приводятся определения и описывается назначение перечисленных диаграмм и моделей применительно к задачам проектирования ПО (в скобках приведены альтернативные названия диаграмм, использующиеся в современной литературе).

Диаграммы прецедентов (диаграммы вариантов использования (use case diagrams)) — это обобщенная модель функционирования системы в окружающей среде.

Диаграммы видов деятельности (диаграммы деятельностей (activity diagrams)) — модель бизнес-процесса или поведения системы в рамках прецедента.

Диаграммы взаимодействия (interaction diagrams) — модель процесса обмена сообщениями между объектами, представляется в виде диаграмм последовательностей (sequence diagrams) или кооперативных диаграмм (collaboration diagrams).

Диаграммы состояний (statechart diagrams) — модель динамического поведения системы и ее компонентов при переходе из одного состояния в другое.

Диаграммы классов (class diagrams) — логическая модель базовой структуры сис-

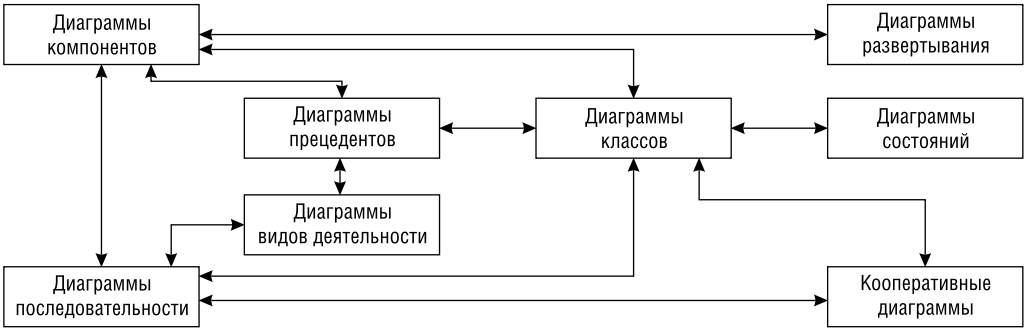

Рис. 1. Взаимосвязи между диаграммами UML

темы, отражающая статическую структуру системы и связи между ее элементами.

Диаграммы базы данных (database diagrams) — модель структуры базы данных, отображающая таблицы, столбцы, ограничения и т. п.

Диаграммы компонентов (component diagrams) — модель иерархии подсистем, отражающая физическое размещение баз данных, приложений и интерфейсов ПО.

Диаграммы развертывания (диаграммы размещения (deployment diagrams)) — модель физической архитектуры системы, отображающая аппаратную конфигурацию ПО.

На рис. 1 показаны отношения между различными видами диаграмм UML. Указатели стрелок можно интерпретировать как отношение «является источником входных данных для ...» (например, диаграмма прецедентов является источником данных для диаграмм видов деятельности и последовательности). Приведенная схема является наглядной иллюстрацией итеративного характера разработки моделей с использованием UML.

Ниже приводятся описания последовательных этапов проектирования ПО с использованием UML.

Разработка модели бизнес-прецедентов

Модель бизнес-прецедентов описывает бизнес-процессы с точки зрения внешнего пользователя, т. е. отражает взгляд на деятельность организации извне.

Проектирование системы начинается с изучения и моделирования бизнес-деятельности организации. На этом этапе вводится и отображается в модели ряд понятий, свойственных объектно-ориентированному подходу.

Исполнитель (действующее лицо (actor)) — личность, организация или система, взаимодействующая с ПО. Различают внешнего исполнителя (который использует или используется системой, т. е. порождает прецеденты деятельности) и внутреннего исполнителя (который обеспечивает реализацию прецедентов деятельности внутри системы).

Прецедент — законченная последовательность действий, инициированная внешним объектом (личностью или системой), которая взаимодействует с ПО и получает в результате некоторое сообщение от ПО.

Класс — описание совокупности однородных объектов с их атрибутами, операциями, отношениями и семантикой.

Ассоциация — связь между двумя элементами модели.

Обобщение — связь между двумя элементами модели, когда один элемент (подкласс) является частным случаем другого элемента (суперкласса).

Агрегация — отношение между элементами модели, когда один элемент является частью другого элемента (агрегата).

Для включения в диаграмму выбранные прецеденты должны удовлетворять следующим критериям:

- прецедент должен описывать, *что* нужно делать, а не *как*;
- прецедент должен описывать действия с точки зрения *исполнителя*;

• прецедент должен возвращать исполнителю некоторое *сообщение*;

• последовательность действий внутри прецедента должна представлять собой одну *неделимую* цепочку.

Выполнение прецедента описывается с помощью диаграмм видов деятельности, которые отображают исполнителей и последовательность выполнения соответствующих бизнес-процессов.

Этап завершается после разработки диаграмм видов деятельности для всех выделенных в модели бизнес-прецедентов. Естественно, на последующих этапах анализа и проектирования будут выявлены какие-то важные подробности в описании деятельности объекта автоматизации. Поэтому разработанные на данном этапе модели будут еще неоднократно корректироваться.

Разработка модели бизнес-объектов

Следующим этапом проектирования информационных систем является разработка модели бизнес-объектов, которая показывает выполнение бизнес-процессов организации ее внутренними исполнителями. Основными компонентами моделей бизнес-объектов являются внешние и внутренние исполнители, а также бизнес-сущности, отображающие все, что используют внутренние исполнители для реализации бизнес-процессов.

Для детального описания выполнения бизнес-процессов обычно используются диаграммы последовательностей.

Результатом этого этапа являются согласованные с заказчиком и достаточно подробные описания действий специалистов организации, внедряющей ПО, необходимые для обеспечения исполнения ее функций.

Разработка концептуальной модели данных

Затем на основе информации, выявленной на этапах бизнес-моделирования, выполняется разработка концептуальной модели данных, которая будет использоваться в разрабатываемой системе.

Этот этап завершает процедуры бизнес-моделирования и позволяет представить команде проектировщиков в едином формате ту информацию, которая будет необходима для создания системы. Разработанные диаграммы являются отправной точкой в процессах проектирования баз данных и приложений системы, обеспечивают согласованность действий бизнес-аналитиков и разработчиков в процессе дальнейшей работы над системой. Эти диаграммы, конечно же, будут претерпевать изменения в процессе последующего проектирования. Однако изменения будут фиксироваться в формате, уже привычном для всей команды разработчиков, и автоматически отражаться в последующих моделях.

Разработка требований к системе

На этапе формирования требований, прежде всего, необходимо определить область действия разрабатываемой системы и получить точное представление о ее желаемых возможностях.

Основой разработки требований является модель системных прецедентов, отражающая выполнение конкретных обязанностей внутренними и внешними исполнителями с использованием ПО.

Источниками данных для создания модели системных прецедентов являются разработанные на предыдущем этапе бизнес-модели. Однако при создании модели полезно предварительно составить детальные описания прецедентов, содержащие определения используемых данных и точную последовательность их выполнения. Описание осуществляется в соответствии с принятым в организации шаблоном, который обычно включает следующие разделы:

• заголовок (название прецедента, ответственный за исполнение, дата создания шаблона/внесения изменений);

• краткое описание прецедента;

• ограничения;

• предусловия (необходимое состояние системы или условия, при которых должен выполняться прецедент);

• постусловия (возможные состояния системы после выполнения прецедента);
• предположения;
• основная последовательность действий;
• альтернативные последовательности действий и условия их инициирующие;
• точки расширения и включения прецедентов.

В процессе создания модели системных прецедентов осуществляется преобразование и перенос компонентов бизнес-моделей на новые диаграммы.

Таким образом, результатом разработки модели системных прецедентов является не только исчерпывающий перечень функций, которые должны быть реализованы в проектируемой системе, но и подробное описание необходимой реализации этих функций.

Анализ требований и предварительное проектирование системы

Основные задачи этапа:

• определить проект системы, который будет отвечать всем бизнес-требованиям;
• разработать общий предварительный проект для всех команд разработчиков (проектировщиков баз данных, разработчиков приложений, системных архитекторов и пр.).

Основными инструментами на данном этапе являются диаграммы классов системы, которые строятся на основе разработанной модели системных прецедентов. Одновременно на этом этапе уточняются диаграммы последовательностей выполнения отдельных прецедентов, что приводит к изменениям в составе объектов и диаграммах классов. Это естественное отражение средствами UML итеративного процесса разработки системы.

Диаграммы классов системы заполняются объектами из модели системных прецедентов. Они содержат описание этих объектов в виде классов и описание взаимодействия между классами.

Таким образом, в результате этапа проектирования появляется достаточно подробное описание состава и функций проектируемой системы, а также информации, которую необходимо использовать в базе данных и в приложениях.

Поскольку диаграммы классов строятся на основе разработанных ранее бизнес-моделей, появляется уверенность в том, что разрабатываемая система будет действительно удовлетворять исходным требованиям заказчика.

В то же время благодаря своему синтаксису, диаграммы классов оказываются хорошим средством структурирования и представления требований к функциональности, интерфейсам и данным для элементов проектируемой системы.

Разработка моделей базы данных и приложений

На этом этапе осуществляется отображение элементов полученных ранее моделей классов в элементы моделей базы данных и приложений. При этом:

• классы отображаются в таблицы;
• атрибуты — в столбцы;
• типы — в типы данных используемой СУБД;
• ассоциации — в связи между таблицами (ассоциации «многие-ко-многим» преобразуются в ассоциации «один-ко-многим» посредством создания дополнительных таблиц связей);
• приложения — в отдельные классы с окончательно определенными и связанными с данными в базе методами и атрибутами.

Поскольку модели базы данных и приложений строятся на основе единой логической модели, автоматически обеспечивается связность этих проектов.

Для каждого простого класса в модели базы данных формируется отдельная таблица, включающая столбцы, которые соответствуют атрибутам класса.

Отображение классов подтипов в таблицах осуществляется одним из стандартных способов:

- одна таблица на класс,
- одна таблица на суперкласс,
- одна таблица на иерархию.

В первом случае для каждого из классов создается отдельная таблица, между которыми затем устанавливаются необходимые связи. Во втором случае создается таблица для суперкласса, а затем в каждую таблицу подклассов включаются столбцы для каждого из атрибутов суперкласса. В третьем — создается единая таблица, содержащая атрибуты как суперкласса, так и всех подклассов. При этом для выделения исходных таблиц подклассов в результирующую таблицу добавляется один или более дополнительных столбцов.

Разработка проекта базы данных осуществляется с использованием специального UML-профиля (Profile for Database Design), который включает следующие основные компоненты диаграмм:

- таблица — набор записей базы данных по определенному объекту;
- столбец — элемент таблицы, содержащий значения одного из атрибутов таблицы;
- первичный ключ (PK) — атрибут, однозначно идентифицирующий строку таблицы;
- внешний ключ (FK) — один или группа атрибутов одной таблицы, которые могут использоваться как первичный ключ другой таблицы;
- обязательная связь — связь между двумя таблицами, при которой дочерняя таблица существует только вместе с родительской;
- необязательная связь — связь между таблицами, при которой каждая из таблиц может существовать независимо от другой;
- представление — виртуальная таблица, которая обладает всеми свойствами обычной таблицы, но не хранится постоянно в базе данных;
- хранимая процедура — функция обработки данных, выполняемая на сервере;
- домен — множество допустимых значений для столбца таблицы.

На диаграммах указываются дополнительные характеристики таблиц и столбцов:

- ограничения — определяют допустимые значения данных в столбце или операции над данными. Ключ (PK, FK) — ограничение, определяющее тип ключа и его столбец; проверка (Check) — ограничение, определяющее правило контроля данных; уникальность (Unique) — ограничение, определяющее, что в столбце содержатся неповторяющиеся данные;
- триггер — программа, выполняющая при определенных условиях предписанные действия с базой данных;
- тип данных и пр.

Результатом этапа является детальное описание проекта базы данных и приложений системы.

Проектирование физической реализации системы

На этом этапе проектирования модели баз данных и приложений дополняются обозначениями их размещения на технических средствах разрабатываемой системы.

Основными понятиями UML, которые используются на данном этапе, являются следующие:

- компонент — самостоятельный физический модуль системы;
- зависимость — связь между двумя элементами, при которой изменения в одном элементе вызывают изменения другого элемента;
- устройство — узел, не обрабатывающий данные;
- процессор — узел, выполняющий обработку данных;
- соединение — связь между устройствами и процессорами.

Диаграммы развертывания позволяют отобразить на единой схеме различные компоненты системы (программные и информационные) и их распределение по комплексу технических средств.

Таким образом, при проектировании ПО она разделяется на части, и каждая из них затем исследуется и создается отдельно. В настоящее время используются два различных способа такого разбиения ПО на подсистемы: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

С позиций проектирования ПО суть функционального разбиения может быть выражена известной формулой: «Программа = = Данные + Алгоритмы». При функциональной декомпозиции программной системы ее структура описывается блок-схемами, узлы которых представляют собой «обрабатывающие центры» (функции), а связи между узлами описывают движение данных.

При объектном разбиении в системе выделяются «активные сущности» — объекты (или компоненты), которые взаимодействуют друг с другом, обмениваясь сообщениями и выполняя соответствующие функции (методы) объекта.

Если при проектировании ПО разбивается на объекты, то для его визуального моделирования следует использовать UML. Если в основу проектирования положена функциональная декомпозиция ПО, то UML не нужен и следует использовать рассмотренные ранее структурные нотации.

В то же время при выборе подхода к разработке ПО следует учитывать, что визуальные модели все более широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. Они хорошо приспособлены для решения таких часто возникающих при создании систем задач, как физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, репликация баз данных, обеспечение безопасности доступа к информационной системе, оптимизация балансировки нагрузки ПО, устойчивость к сбоям и т. п. Визуализированные средствами UML модели ПО позволяют наладить плодотворное взаимодействие между заказчиками, пользователями и командой разработчиков. Они обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте.

Визуальное проектирование на основе SysML

Обычно расширение UML выполняется путем введения дополнительных стереотипов с определенной семантикой, что обеспечивает представление артефактов той предметной области, для которой требуется такое расширение. Однако для моделирования аппаратуры этого оказалось недостаточно. Понадобилось добавить ряд новых графических элементов и диаграмм, которые позволяют описывать нюансы каждого элемента модели и взаимосвязей между элементами, а также строго задавать границы модели. Кроме того, в рамках поставленной задачи UML характеризуется некоторой избыточностью, поэтому не все его элементы вошли в новый экземпляр.

Изменения были сформулированы в виде профиля UML 2.0 и названы SysML. В спецификацию этого языка вошли новые диаграммы: требований, внешних и внутренних блоков, времени, параметрическая. Сохранены следующие диаграммы UML: вариантов использования, конечных автоматов (в прежних версиях ее называли диаграммой состояний), деятельности и последовательности.

Формализация требований к проектируемой системе

Для описания требований к проектируемой системе и определения взаимосвязей между ними в SysML используют классическую UML-диаграмму вариантов использования, собственную диаграмму требований и различные параметрические ограничения.

При разработке модели на языке SysML важно отслеживать ее соответствие требованиям к разрабатываемому продукту, т. е. обеспечивать их трассировку. Эта задача осложняется тем, что на протяжении жиз-

ненного цикла проекта для описания требований используют разнообразные форматы, инструменты разной сложности и функциональности (*Excel*, *Adobe FrameMaker*, *Telelogic DOORS*, *IBM Rational RequisitePro* и др.). Небольшое число требований еще поддается управлению вручную, но когда они исчисляются сотнями или тысячами нужны специальные инструменты. При выборе инструмента управления следует определить позволяет ли он организовывать сложные наборы требований из нескольких источников (формировать из них единую связанную среду управления) и создавать предназначенные для наглядного анализа отчеты о влиянии требований на модель. Впрочем, вопрос трассировки требований относится не столько к языку SysML, сколько к поддерживающим его инструментам.

Описание структуры системы

После того как требования к разрабатываемой системе специфицированы, моделируется ее структура. Для этой цели в SysML вместо классических UML-диаграмм (классов, объектов и др.) используются диаграммы внутренних и внешних блоков, т. е. модульных единиц, инкапсулирующих атрибуты, операции и ограничения. Можно сказать, что блок является своего рода расширением объекта.

Диаграммы блоков и расширяющие их параметрические диаграммы обеспечивают SysML-модели четкость структуры, необходимую для моделирования аппаратных и программно-аппаратных средств. А поскольку SysML устанавливает строгие правила, которым должна соответствовать модель, появляется возможность эффективного статического анализа моделей: их проверки для обнаружения семантических и синтаксических ошибок, противоречий и конфликтов. Среди проблем, выявление которых обеспечивает статический анализ, можно указать следующие: неразрешенные, циклические и пустые ссылки, неподходящие интерфейсы или параметры, передаваемые одним элементом другому, отсутствие сопроводительной информации (например, описания блока), ссылки на несуществующие элементы или не имеющие ссылок элементы.

Некоторые из проблем могут иметь катастрофические последствия на этапе реализации модели, но обнаружить их удается лишь с помощью исчерпывающей проверки либо после завершения реализации. Инструменты, обеспечивающие полномасштабный статический анализ, дают пользователю возможность поддерживать корректность и целостность модели, выполнять одну или несколько проверок в любой момент. Кроме того, необходимо, чтобы пользователи могли сами определять методы проверки модели, например на основе отраслевых стандартов или стандартов организации.

Полномасштабный статический анализ модели помогает команде разработчиков избежать дорогостоящих ошибок на этапе моделирования и убедиться в том, что все участники проекта адекватно понимают стоящие перед ними задачи.

Описание и имитация поведения системы

Для описания поведенческих аспектов проектируемой системы в SysML используются обычные диаграммы UML, такие как диаграммы конечных автоматов.

Процесс проектирования

Необходимо помнить, что SysML как и UML, — это язык моделирования, а не методология. Порядок применения SysML на практике определяет процесс, например *MSF* (*Microsoft Solutions Framework*) или *XP* (*Extreme Programming*).

Список литературы

1. *Брауде Э.* Технология разработки программного обеспечения. СПб.: Питер, 2004.

2. *Буч Г.*, *Якобсон А.*, *Рамбо Дж.* UML. Классика CS. Изд. 2-е / Пер. с англ.; под общей редакцией проф. С. Орлова. СПб.: Питер, 2005.

3. *Рамбо Дж.*, *Блаха М.* UML 2.0. Объектно-ориентированное моделирование и разработка. Изд. 2-е. СПб.: Питер, 2007.