Содержание

Предисловие

Предисловие Питера Фримена

Недавние успехи в области разработки аппаратного обеспечения и средств связи привели к лавинообразному росту числа параллельных и распределенных систем, а также систем реального времени. Это, в свою очередь, способствовало изменению требований, предъявляемых к программному обеспечению. Широкое распространение объектно-ориентированных методов, а теперь еще и применение языка UML, разумеется, влияют на сложившуюся практику, однако темпы, к сожалению, отстают от требований времени.

Одна из причин такого отставания – отсутствие добротных практических руководств по объектно-ориентированному анализу и проектированию параллельных приложений (в особенности таких, которые одновременно являются распределенными и/или должны работать в режиме реального времени). Книга, которую вы держите в руках, призвана восполнить этот пробел.

Я не знаю человека, который мог бы написать авторитетный учебник по данному предмету лучше, чем Хассан Гома. Более 20 лет Хассан изучал природу систем реального времени, параллельных и распределенных, работая проектировщиком, проводя исследования в области новых методов проектирования подобных систем и преподавая в университете. Результаты его деятельности перед вами.

Книга прекрасно организована, поскольку написал ее опытный преподаватель. В ней продемонстрирована такая глубина понимания технологии, которая появляется только в результате длительных исследований в данной области. А рисунки и практические рекомендации свидетельствуют о наличии у автора непосредственного опыта проектирования программных систем.

Надеюсь, что вы получите от чтения этой книги такое же удовольствие, как и я, и будете пользоваться ею еще много лет.

Предисловие Брана Селика

> В условиях недостаточности данных
> допускается куда меньше ошибок,
> чем при полном отсутствии данных.
>
> *Чарльз Бэббидж*

Недавно, произведя поиск на Web-сайте, специализирующемся в области книготорговли, я обнаружил 1188 названий в категории «проектирование программного обеспечения», и число книг на эту тему постоянно растет. Несмотря на такое

изобилие, в большинстве подобных изданий почти ничего не говорится собственно о теории и практике проектирования, а потому огромное количество программ в наши дни создается людьми, плохо знакомыми с предметом.

В традиционных инженерных дисциплинах всегда использовались последние достижения прикладной математики, гарантирующие, что проект будет отвечать поставленным требованиям при приемлемых затратах. Основываясь на оценках, полученных с помощью математической модели, можно достаточно уверенно приступать, например, к конструированию моста. Однако применительно к программному обеспечению проектирование оказывается преимущественно неформальным процессом, для которого зачастую нет моделей и методов прогнозирования результата. С этой точки зрения весьма поучительно было бы сравнить эволюцию программного и аппаратного обеспечения на протяжении нескольких последних десятилетий. Если аппаратные устройства со временем становились миниатюрнее, быстрее и дешевле, то программы, напротив, оказывались все более объемными, медленными, дорогими и менее надежными. Одна из основных причин такого положения состоит в том, что в основе проектирования современной аппаратуры лежит использование прогностических моделей.

Отсутствие фундаментальных инженерных принципов в практике разработки ПО можно отчасти объяснить изменчивой, в чем-то даже хаотичной природой программ, что сильно затрудняет математическое моделирование. Тем не менее имеется немало весьма полезных аналитических методик. Особенно активно такие методики развивались в области создания систем реального времени: здесь надежное предсказание временных характеристик программы часто оказывается критически важным, поскольку от этого зависит человеческая жизнь. И все же, вопреки доказанной на опыте эффективности, подобные методы распространены не очень широко. По сути дела, многие разработчики систем реального времени даже не подозревают об их существовании.

Написание программы – это в основном интеллектуальное упражнение, не ограниченное такими физическими факторами, как необходимость обрабатывать сырье и затрачивать много энергии. У большинства практикующих программистов, соблазненных кажущимся отсутствием сопротивления, создается впечатление, что важен только сам код, а потому они не разделяют процессы проектирования и кодирования. Как это ни странно, те же самые люди прекрасно понимают разницу между проектированием и сборкой авиалайнера.

Дополнительным препятствием к внедрению вышеупомянутых методик в практику создания ПО служит то, что некоторые из них ориентированы на традиционное процедурное программирование. Хотя никаких фундаментальных зависимостей от такой модели не имеется, проблема привязки к более современной объектно-ориентированной модели все же существует, и тем, кто ценит преимущества, заложенные в новой парадигме, это мешает пользоваться старыми методами проектирования.

Книга Хассана Гома – первая из известных мне, в которой эти вопросы рассматриваются систематически и всесторонне. Она представляет собой не собрание разнородных паттернов и методик, а ясное и подробное изложение способов

объединения традиционных методов инженерного проектирования с унифицированным языком моделирования (UML), который уже обрел статус стандарта. Кроме того, демонстрируется роль данных методов в процессе создания конкретных систем: параллельных, распределенных и реального времени. (Опытные разработчики знают, какие трудности стоят за каждым из этих терминов по отдельности, а создание систем, обладающих всеми тремя характеристиками, – одна из наиболее сложных инженерных задач.)

В соответствии с хорошо известным принципом, что обучаться лучше всего на практике, значительную часть книги занимают детально продуманные нетривиальные примеры. Вам будет очень полезно полностью разобраться хотя бы в одном из них, тогда вы по-настоящему оцените и удобство предлагаемого подхода, и то, как на самом деле должно выглядеть инженерное проектирование программного обеспечения.

Нотация UML и методы проектирования программ

В этой книге рассматривается объектно-ориентированный анализ и проектирование параллельных приложений, в частности распределенных систем и систем реального времени. Объектно-ориентированные концепции играют огромную роль в анализе и проектировании программного обеспечения, поскольку касаются фундаментальных вопросов адаптируемости и развития. На протяжении некоторого времени существовало множество разнообразных нотаций и методов объектно-ориентированного анализа и проектирования программных систем, но потом появился унифицированный язык моделирования (UML), который предоставляет стандартизованную нотацию для описания таких моделей. Однако UML лучше использовать в сочетании с методами объектно-ориентированного анализа и проектирования.

В большинстве изданий, посвященных объектно-ориентированному анализу и проектированию, рассматриваются только последовательные системы, а важные вопросы, которые возникают в связи с разработкой распределенных систем и систем реального времени, опускаются. Для создания такого рода приложений нужно объединить объектно-ориентированные концепции с методами параллельной обработки. Поскольку на сегодняшний день стандартной нотацией для описания подобных моделей является UML, то именно этим языком мы и будем пользоваться далее.

COMET – метод архитектурного проектирования и моделирования параллельных объектов

COMET (Concurrent Object Modeling and Architectural Design Method) – это метод разработки параллельных приложений, в частности распределенных систем и систем реального времени. В основе создания объектно-ориентированного ПО по методу COMET лежит концепция прецедентов, а его жизненный цикл характеризуется большим числом итераций. На этапе моделирования требований

(Requirements Modeling) система рассматривается как черный ящик. Формируется модель прецедентов, где определяются функциональные требования к системе в терминах актеров и прецедентов.

На этапе аналитического моделирования (Analysis Modeling) строятся статическая и динамическая модели системы. Статическая модель описывает структурные отношения между классами предметной области. Для выявления объектов, рассматриваемых в аналитической модели, применяется критерий разбиения на объекты. После этого разрабатывается динамическая модель и уточняются описанные в модели требований прецеденты с целью представить объекты, участвующие в каждом прецеденте, и взаимодействия между ними. В динамической модели с помощью диаграмм состояний определяются объекты, зависящие от состояния.

На этапе проектного моделирования (Design Modeling) продумывается архитектура системы. Аналитическая модель, в которой основное внимание уделялось предметной области, соотносится со средой, где будет эксплуатироваться программа, и с проектной моделью, где акцент ставится на область решения. Формулируются критерии разбиения системы на подсистемы. В случае распределенной системы наиболее важным является разделение ответственности между клиентами и серверами, в том числе с точки зрения централизации и распределения данных и управления. Кроме того, проектируются интерфейсы для обмена сообщениями, рассматриваются синхронные, асинхронные, групповые коммуникации и брокерские сервисы. Затем наступает черед проектирования отдельных подсистем. Проектирование параллельных приложений, в том числе и систем реального времени, сводится в основном к выделению параллельно выполняемых объектно-ориентированных задач. Создаются интерфейсы для обмена данными между задачами и синхронизации. При анализе производительности новой системы реального времени используется метод монотонного анализа частот, предложенный Институтом проектирования программных систем (Software Engineering Institute).

Особенности книги

Существует несколько книг, описывающих концепции и методы объектно-ориентированного анализа для приложений общего вида. Однако распределенные системы и системы реального времени имеют специфические особенности, которые в большинстве изданий рассматриваются лишь поверхностно. Здесь вы найдете исчерпывающие сведения о том, как фундаментальные объектно-ориентированные концепции применяются к анализу и проектированию распределенных программ (включая клиент-серверные приложения) и программ реального времени. Помимо таких объектно-ориентированных концепций, как сокрытие информации, классы и наследование, в этой книге рассказывается о конечных автоматах, параллельных задачах, технологии распределенных объектов и планировании в режиме реального времени. Подробно описывается метод COMET, представляющий собой специализацию основанного на UML подхода к объектно-ориентированному анализу и проектированию параллельных и распределенных систем, а также систем реального времени. Чтобы продемонстрировать применение COMET на

практике, в издание включено несколько примеров из разных областей: система реального времени, система клиент-сервер и распределенная система.

Отличительные черты данной книги состоят в следующем:

- здесь рассказывается о критериях разбиения, призванных помочь программисту выявлять подсистемы, объекты и параллельно выполняемые задачи на разных стадиях процесса анализа и проектирования;
- с помощью динамического моделирования взаимодействия объектов и конечных автоматов описываются способы совместного использования диаграмм кооперации и диаграмм состояний;
- делается акцент на параллельности – приводятся характеристики активных и пассивных объектов;
- рассматриваются проектирование распределенных приложений и методы взаимодействия распределенных компонентов;
- анализируется производительность системы реального времени с помощью теории планирования в реальном времени;
- приводятся развернутые примеры различных приложений, иллюстрирующие применение изложенных концепций и методов.

Структура книги

Книга состоит из трех частей. Первая часть содержит обзор различных концепций, технологий, жизненных циклов и методов проектирования систем реального времени, параллельных и распределенных систем. Глава 1 начинается с краткого описания различий между методом и нотацией, за которым следует обсуждение характеристик распределенных систем и систем реального времени. В главе 2 излагаются те аспекты нотации UML, которые задействованы в методе COMET. Далее рассматриваются важные концепции проектирования (глава 3) и необходимой технологической поддержки (глава 4) для параллельных и распределенных систем. Затем в главе 5 рассказывается о жизненных циклах ПО и методах проектирования.

Во второй части речь идет о методе COMET. В главе 6 анализируется жизненный цикл ПО, созданного с использованием COMET. В главе 7 говорится об этапе моделирования требований в COMET и, в частности, о моделировании прецедентов. В главах 8–11 описываются этапы аналитического моделирования в COMET. Главы 12–16 посвящены этапу проектного моделирования. Предмет главы 17 – расчет производительности созданной системы с помощью теории планирования в реальном времени и метода монотонного анализа частот.

В третьей части метод COMET демонстрируется на пяти подробных примерах проектирования распределенных приложений (две системы реального времени, система клиент-сервер и две распределенные системы). В главе 18 рассматривается система управления лифтами и предлагаются два решения: одно распределенное, другое – нет. В главе 19 исследуется банковская клиент-серверная система. Глава 20 посвящена автомобильной системе круиз-контроля, глава 21 – распределенной системе автоматизации производства, а глава 22 – распределенной системе электронной коммерции.

Как работать с книгой

Эту книгу можно читать по-разному. Допустимо читать подряд: главы 1–5 являются введением в используемые концепции и технологии, в главе 6 приводится обзор метода COMET, главы 7–17 посвящены углубленному изучению порядка проектирования приложений с помощью COMET, а главы 18–22 включают примеры и их анализ.

Первая часть – вводная. Если вы владеете предметом, можете пропустить ее и сразу перейти к описанию метода COMET во второй части. Тем, кто знаком с языком UML, не понадобится глава 2. Зная концепции проектирования ПО, вы не станете читать главу 3. Если вы обладаете сведениями о технологиях, применяемых в параллельных и распределенных системах, пропустите главу 4; если же вам известны жизненные циклы ПО и методы проектирования, пропустите главу 5. Читателям, которые заинтересованы главным образом в методе COMET, удобнее сразу перейти ко второй и третьей частям книги. Тем, чья работа связана с проектированием распределенных приложений, следует прочесть главы 4, 12 и 13 (дополнительная информация о параллельных системах приводится в главах 14–16), а также изучить примеры распределенных приложений в главах 18, 19, 21 и 22. Желающим познакомиться с проектированием систем реального времени и теорией планирования в реальном времени необходимо прочесть главы 4, 14–17 и разобрать примеры в главах 18 и 20.

Опытные проектировщики могут использовать это издание в качестве справочника, обращаясь к тем или иным главам на разных этапах анализа или проектирования проекта. Главы мало связаны друг с другом. Например, краткое описание прецедентов вы найдете в главе 7, при проектировании диаграмм состояний можете заглянуть в главу 10, а при разработке динамической модели – в главу 11, проектирование распределенных компонентов рассматривается в главе 13, проектирование параллельно выполняемых задач – в главе 14, а информацию о планировании в реальном времени вы найдете в главе 17. Разобраться в том, как используется метод COMET, несложно и путем изучения примеров, поскольку в каждом из них объясняются мотивы принятия решений на каждом этапе проектирования.

Благодарности

Я очень признателен рецензентам ранних редакций рукописи. Особая благодарность Джеффу Мейджи (Jeff Magee), Ларри МакАлистеру (Larry McAlister), Кевину Миллсу (Kevin Mills), Роберту Дж. Петтиту IV (Robert G. Pettit IV) и Марии Эрикссон (Maria Ericsson) за проницательные замечания. Хочу также поблагодарить Антуана К. Дина (Antuan Q. Dinh), Гулама Ахмада Фаруха (Ghulam Ahmad Farukh), Йохана Галле (Johan Galle), Келли Хьюстона (Kelli Houston), Джишну Мукерджи (Jishnu Mukerji), Лесли Пробаско (Leslee Probasco), Санджива Сетия (Sanjeev Setia) и Думинду Виджесекера (Duminda Wijesekera) за ценные рецензии.

Спасибо Кевину Миллсу за вклад в использование стереотипов в COMET, Шигеру Оцуки (Schigeru Otsuki) – за помощь в работе над материалом по паттернам проектирования, Роджеру Александру (Roger Alexander) – за участие в работе над одним из примеров главы 15, Ларри МакАлистеру, помогавшему сделать рис. 21.1. Особая благодарность – Тиррелл Элбо (Tyrrell Albaugh) за самоотверженный труд по координации процесса выпуска книги, Кристин Эрикссон (Kristin Ericsson) – за организацию издательского процесса и Мелинде МакКейн (Melinda McCain) – за тщательную корректуру рукописи.

Я признателен студентам, принимавшим участие в семинаре по проектированию программных систем в Университете Джорджа Мейсона, за энтузиазм, преданность и ценные замечания, а также Институту проектирования программных систем (Software Engineering Institute – SEI) за предоставленный материал по планированию в реальном времени, частично положенный в основу главы 17. Необходимо поблагодарить и Консорциум по повышению производительности разработки ПО (Software Productivity Consortium) за спонсорскую помощь, оказанную при работе над первой редакцией глав 9–11 этой книги. Спасибо Арману Анвару (Arman Anwar), Хуа Линь (Hua Lin) и Майклу Шину (Michael Shin) за изготовление ранних версий рисунков.

И, разумеется, я хочу поблагодарить свою жену Джилл за понимание, ободрение и поддержку.

Типографские соглашения и альтернативная нотация

В этом разделе описываются использованные в книге типографские соглашения. Здесь же приводится альтернативная нотация UML для обозначения стереотипов и активных объектов.

Принятые обозначения

Для улучшения восприятия соглашения, применяемые для записи имен классов, объектов и т.д. на рисунках, иногда отличаются от записи тех же имен в тексте. На рисунках имена набраны шрифтом Times Roman, а в тексте – шрифтом `Courier` (этим же моноширинным шрифтом в книге набраны листинги – фрагменты программного кода). Основные термины в тексте выделены *курсивом*, нотации UML отмечены **полужирным шрифтом**. Используемые обозначения во многом определяются этапом проектирования. Заглавные буквы в названиях употребляются по-разному в зависимости от того, об аналитической (менее формальной) или проектной (более формальной) моделях идет речь.

Моделирование требований

Имена прецедентов набраны шрифтом `Courier`, первые буквы каждого слова в имени – заглавные, например `Снять Деньги`.

Аналитическая модель

Ниже приведены соглашения, применяемые в аналитических моделях.

Классы

Имена классов набраны шрифтом `Courier` с начальными заглавными буквами. На рисунках отдельные слова в именах классов не разделяются пробелами (`ЧековыйСчет`), а в тексте – разделяются для удобства чтения (`Чековый Счет`).

Имена атрибутов начинаются с маленькой буквы, например `баланс`. Если имя атрибута состоит из нескольких слов, то на рисунках они не разделяются пробелами, а в тексте – разделяются. Первое слово имени начинается с маленькой буквы, остальные – с большой, допустим `номер Счета`.

Тип атрибута пишется с большой буквы, скажем `Boolean`, `Integer` или `Real`.

Объекты

Имена объектов записываются по-разному. На рисунках, в отличие от текста, они всегда подчеркиваются. Кроме того, применяются следующие соглашения:

- отдельный именованный объект. В этом случае первая буква в первом слове имени – маленькая, а в последующих словах – большая. На рисунках имя объекта выглядит так: `чековыйСчет` или `другойЧековыйСчет`. В тексте те же объекты обозначаются иначе: `чековый Счет` или `другой Чековый Счет`;
- отдельный неименованный объект. Иногда объекты на рисунках показаны как неименованные экземпляры класса, например `:ЧековыйСчет`. В тексте такой объект будет обозначен как `Чековый Счет`. Для удобства восприятия двоеточие опускается, а между словами в составном имени вставляются пробелы.

Таким образом, в зависимости от способа изображения объекта на рисунке его имя в тексте иногда начинается с большой, а иногда с маленькой буквы.

Сообщения

В аналитической модели все сообщения являются простыми (см. рис. 2.11), поскольку решение об их типе еще не принято. Имена сообщений начинаются с большой буквы. Если имя состоит из нескольких слов, то они разделяются пробелами, например `Имя Простого Сообщения`.

Обозначения на диаграммах состояний

Имена состояний, событий, условий, действий и деятельностей начинаются с большой буквы, слова в составном имени разделяются запятыми, например состояние `Ожидание ПИН-кода`, событие `Наличные Выданы` и действие `Выдать Наличные`.

Проектная модель

Ниже приведены соглашения, применяемые в проектных моделях.

Активные и пассивные классы

Соглашения об именовании активных и пассивных классов такие же, как для классов в аналитической модели.

Активные и пассивные объекты

Соглашения об именовании активных и пассивных объектов такие же, как для объектов в аналитической модели.

Сообщения

В проектной модели первое слово в имени сообщения начинается с маленькой буквы, остальные – с большой. На рисунках слова не разделяются пробелами (`тревожноеСообщение`), а в тексте пробелы между ними есть (`тревожное Сообщение`).

Имена параметров сообщений начинаются с маленькой буквы, например `скорость`. Если имя состоит из нескольких слов, то на рисунках оно записывается без разделительных пробелов, а в тексте – с ними. Первое слово в составном имени начинается с маленькой буквы, остальные – с большой, скажем `полныйПробег` (на рисунке) или `полный Пробег` (в тексте).

Альтернативная нотация для стереотипов

В UML нотация, принятая для стереотипов, позволяет адаптировать элементы модели к особенностям конкретной предметной области. С одной стороны, стереотипы можно обозначать с помощью двойных угловых кавычек внутри элемента модели (например, класса или объекта), как показано на рис. 1а. С другой стороны, в UML допускается изображение стереотипа в виде графического символа. Наиболее распространенные на сегодняшний день пиктограммы были введены Джекобсоном [Jacobson 1992] и применяются в унифицированном процессе разработки программ [Jacobson, Booch, and Rumbaugh 1999] на этапе аналитического моделирования. Стереотипы используются для представления классов-сущностей (entity), граничных классов (boundary, в методе COMET им соответствуют классы со стереотипом «интерфейс») и управляющих классов (control). На рис. 1б с помощью графических стереотипов Джекобсона изображены сущностный класс

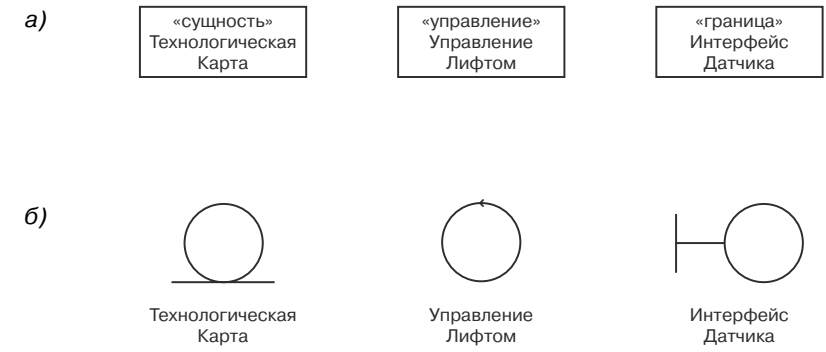

Рис. 1. Альтернативные нотации для стереотипов в UML:
а – стандартная нотация UML для стереотипов; б – нотация для стереотипов, применяемая в унифицированном процессе разработки программ

Технологическая Карта, управляющий класс Управление Лифтом и граничный класс Интерфейс Датчика.

Альтернативная нотация для активных объектов

В этой книге применяется стандартная нотация UML для обозначения активных (в методе COMET они называются задачами) и пассивных объектов. В UML прямоугольник активного объекта рисуется с жирной рамкой, а прямоугольник пассивного объекта – с тонкой (см. главу 2). Некоторые CASE-средства не поддерживают такой стандарт, поэтому активные и пассивные объекты оказываются визуально неразличимыми. В COMET слово «задача» обычно не употребляется в имени стереотипа активного объекта. Как показано на рис. 2а на примере объекта Генератор Отчетов, для этой цели достаточно прямоугольника с жирной рамкой. Если же такой визуальный эффект не поддерживается, то в имя стереотипа можно вставить слово «задача» (см. рис. 2б).

Рис. 2. Альтернативные нотации для активных объектов (задач) в UML: а – стандартная нотация UML для активных объектов; б – вариант, используемый, если стандартная нотация активных объектов не поддерживается

ЧАСТЬ I

Нотация UML, концепции проектирования, технологии, жизненные циклы и методы

Глава 1. Введение

Значительное понижение цен на микропроцессоры и полупроводниковые микросхемы и столь же существенное увеличение производительности микропроцессоров, наблюдаемые на протяжении нескольких лет, сделали рентабельными распределенные системы и системы реального времени на базе микрокомпьютеров. Сегодня большинство коммерческих, промышленных, военных, медицинских и потребительских продуктов снабжаются микропроцессорами и целиком либо в значительной части управляются программами. Такие системы встречаются в микроволновых печах и видеомагнитофонах, в телефонах и телевизорах, в автомобилях и самолетах, в подводных лодках и космических кораблях, в автоматах по продаже газированных напитков и банкоматах, в системах диагностики пациентов и системах управления производством, в контроллерах роботов и системах управления лифтами, в системах управления городским и воздушным транспортом, в электронной почте и коммерции, в «интеллектуальных» транспортных и информационных магистралях... Список можно продолжать до бесконечности. Все это параллельные системы, а многие из них являются к тому же распределенными или системами реального времени.

Объектно-ориентированные концепции особенно важны для анализа и проектирования программного обеспечения, поскольку они касаются фундаментальных вопросов адаптируемости и развития. Сравнительно недавно появившийся унифицированный язык моделирования (UML) предлагает стандартизованную нотацию для описания объектно-ориентированных моделей [Booch, Rumbaugh, Jacobson 1998; Eriksson and Penker 1998; Fowler and Scott 1999; Jacobson, Booch and Rumbaugh 1999; Rumbaugh, Booch and Jacobson 1999]. Однако в большинстве книг, посвященных данным темам, рассматриваются только последовательные приложения и опускаются существенные вопросы, связанные с проектированием систем реального времени, параллельных и распределенных систем. Объединение концепций объектно-ориентированного проектирования с концепциями параллельного выполнения необходимо для успешного создания распределенных приложений, работающих в реальном масштабе времени. Поскольку UML содержит стандартную нотацию для описания объектно-ориентированных моделей, в книге будет использоваться именно этот язык. Особое внимание уделяется моделированию динамики системы, представляющему интерес для приложений реального времени и распределенных приложений.

В книге рассматривается объектно-ориентированный анализ и проектирование параллельных систем с использованием нотации UML. В частности, описываются две важнейшие категории параллельных приложений: распределенные

и реального времени. Издание предназначено для тех, кто хочет научиться проектировать и оценивать программное обеспечение параллельных и распределенных систем и систем реального времени.

В этой главе вводятся объектно-ориентированные концепции и обсуждаются общие характеристики трех вышеупомянутых категорий программных систем.

1.1. Объектно-ориентированные методы и UML

Объектно-ориентированные методы основаны на концепциях сокрытия информации, классов и наследования [Wegner 1990]. *Сокрытие информации* [Parnas 1972, Parnas 1979] позволяет получить замкнутые, а оттого в большей степени поддающиеся модификации и сопровождению системы. *Наследование* [Meyer 1987, Meyer 1997, Wegner 1990] – это систематический способ адаптации классов.

Язык UML, пришедший на смену многочисленным системам нотации и методикам проектирования [Booch 1994; Coad and Yourdon 1991; Jacobson 1992; Rumbaugh et al. 1991; Shlaer and Mellor 1988; Wirfs-Brock, Wickerson, and Wiener 1990 и др.], предложил нотацию для описания объектно-ориентированных моделей, которая стала промышленным стандартом. Однако для эффективного применения нотации UML необходимо сочетать ее с каким-либо методом объектно-ориентированного анализа и проектирования.

В описываемом методе COMET сочетаются прецеденты использования [Jacobson 1992], статическое моделирование [Rumbaugh et al. 1991, Booch 1994], диаграммы состояний [Harel 1988, Harel and Politi 1998, Coleman et al. 1994, Rumbaugh et al. 1991] и диаграммы последовательности событий, которые встречаются в нескольких методах [Booch 1994, Gomaa 1993, Jacobson 1992, Rumbaugh et al. 1991]. Применяемая нотация основана на UML [Booch, Rumbaugh, and Jacobson 1998; Eriksson and Penker 1998; Fowler and Scott 1999; Jacobson, Booch and Rumbaugh 1999; Rumbaugh, Booch and Jacobson 1999]. В ходе *моделирования прецедентов использования* определяются функциональные требования к системе в терминах актеров и прецедентов. *Статическая модель* предлагает статический взгляд на информационные аспекты системы. Класс определяется в терминах своих атрибутов и взаимоотношений с другими классами. Результатом *динамического моделирования* является динамический взгляд на систему. Уточняются сформулированные ранее прецеденты с целью показать взаимодействие объектов, участвующих в каждом из них. Разрабатываются диаграммы кооперации и последовательности, отражающие кооперацию объектов в каждом прецеденте. Зависящие от состояния аспекты системы описываются с помощью диаграмм состояний, причем для каждого объекта составляется своя диаграмма.

В аналитической модели основное внимание уделяется пониманию проблемы: выявлению объектов предметной области и передаваемой между ними информации. Объекты и классы предметной области группируются на основе критериев выделения объектов. Рассмотрение вопросов, активен объект или пассивен, синхронно или асинхронно посылаемое сообщение и какая вызывается операция у объекта-получателя, откладывается до стадии проектирования.

На этапе проектирования среди объектов выявляются активные (их называют задачами) и пассивные (их называют объектами). Для определения задач применяются критерии разбиения на задачи. Разрабатываются также интерфейсы задач и описываются операции каждого пассивного класса.

1.2. Метод и нотация

Метод проектирования и нотация проектирования – это разные вещи. Нотация проектирования ПО предназначена для описания самого проекта. Хотя она и предполагает наличие определенного подхода к проектированию, сам подход остается за ее рамками. Метод проектирования ПО представляет собой систематическое описание этапов создания проекта.

Нотация проектирования ПО описывает проект программы в графическом или текстовом виде. В частности, диаграммы классов – это графическая нотация, а псевдокод – текстовая.

Концепция проектирования ПО – это фундаментальная идея, применимая к проектированию всей системы, например сокрытие информации.

Стратегия проектирования ПО – общий план и методика выполнения проекта. Одной из стратегий является объектно-ориентированная декомпозиция.

Критерии структурирования ПО – это эвристические или формальные правила, помогающие проектировщику разбить систему на отдельные компоненты. Так, критерии разбиения на объекты – это правила декомпозиции системы на объекты.

Метод проектирования ПО описывает последовательность шагов, выполняемых при работе над проектом при условии, что требования к системе уже сформулированы. Он помогает выявить, какие решения предстоит принять, в каком порядке это следует делать и на основе каких критериев. Метод проектирования базируется на наборе соответствующих концепций, использует одну или несколько стратегий, а также ту или иную нотацию для документирования результатов. При выполнении определенных шагов метод может подсказать разработчику, какие критерии наиболее удобны для декомпозиции системы.

В методе COMET для описания проекта применяется нотация UML. Метод основан на следующих концепциях: сокрытие информации, наследование и параллельные задачи. Стратегия проектирования параллельно работающих объектов состоит в разбиении системы на активные и пассивные объекты и определении интерфейсов между ними. Метод COMET также содержит критерии разбиения, помогающие выделить объекты в системе на этапе анализа, а затем на этапе проектирования выявить отдельные подсистемы и параллельно выполняемые задачи.

1.3. Параллельные приложения

На заре развития вычислительной техники компьютерные приложения являлись в основном пакетными заданиями. Все программы были последовательными и работали в автономном режиме. Теперь же, когда имеется огромное число