18 марта 2019г.

Лекция № 3

Тема: «Проектирование программных приложений баз данных в СОД по этапам методологии проектирования ИС»

Наличие огромного количества программных компонентов, сервисов и информационных систем смещает процесс разработки информационных систем от задач программирования к задачам выбора компонентов и моделирования их совместного взаимодействия. При этом многие существующие методологии используют модельно-ориентированные подходы к проектированию и графические языки моделирования, представляющие модели в виде набора диаграмм. Каждый этап проектирования описывается различными моделями с отличающимися семантиками и способами формализации, видами деятельности и предполагает наличие соответствующих специальных инструментальных средств. Отсутствие единого формального подхода препятствует совместному использованию моделей, их интерпретации, применению методов автоматического сквозного согласования моделей на всех этапах жизненного цикла информационной системы. Таким образом, имеются проблемы как в области формализации описаний систем, так и в области согласования семантик и нотаций, а также совместного интерпретирования моделей.

Отсутствие единого формального подхода проектирования СОД приводит к мерам применения методологии разработки ИС к проектированию СОД

В существующих методологиях разработка ИС трактуется как процесс отображения функциональных требований (модель «черного ящика»), предъявляемых к системе, в некоторую организацию конструкционных элементов, реализующую требуемые функции (модель «белого ящика»).

Согласование группы систем возможно в рамках системного подхода, выделяя в качестве системообразующих несколько факторов, таких как — архитектура системы, требования заинтересованных лиц и подхода стандарта ISO 42010 (Рекомендованная практика архитектурного описания программоемких систем). В этом стандарте архитектура системы рассматривается как согласование архитектурных описаний, учитывающих интересы (concerns) заинтересованных сторон (stakeholders). Архитектурное решение представлено моделями, описывающими связь функции системы и ее конструкции. Каждая группа описаний (ГО), включающая набор моделей, раскрывает отдельный аспект системы, а набор групп образует ее полное описание. Соглашения, по которым ГО создается, отображаются и анализируются, устанавливаются методом описания (viewpoint).

Стандарт не определяет способа согласования архитектурных описаний. Для их согласования введём семантический уровень представления архитектурных описаний, в рамках которого будем рассматривать все архитектурные описания единообразно, как некоторые знания и использовать для согласования знаний методы управления знаниями [94].

Многие современные информационные системы являются программными приложениями баз данных. При этом в качестве хранилища информации используются реляционные системы управления базами данных (РСУБД), а для реализации бизнес-логики применяют объектно-ориентированные языки программирования (ООЯП). Из-за различий в подходах к управлению данными возникает так называемое объектно-ориентированное несоответствие (ОРН), для преодоления последствий которого применяется множество частных решений. В данной главе описана разработанная автором методология метамодельного проектирования приложений баз данных (МММППБД), которая позволяет выполнить различные этапы проектирования разрабатываемой информационной системы, и в конечном итоге получить приложение с развитым графическим интерфейсом. Описанная методология построена на основе архитектурных решений, подробно представленных в предыдущих главах, и содержит все преимущества, о которых написано ранее.

Данная методология разрабатывалась автором в течение последних нескольких лет и протестирована на множестве проектов в различных сферах деятельности: от высокочастотного биржевого трейдинга, автоматизированного управления сетью салонов красоты и до управления крупным агрохолдингом. МММППБД подразумевает следующие этапы:

1. Концептуальное проектирование;
2. Метамодельное проектирование;
3. Сущностное проектирование;
4. Имплементационное проектирование;
5. Презентационное проектирование.

Концептуальное проектирование также называют инфологическим и используют для построения модели предметной области, применяя при этом различные графические нотации. В настоящее время очень часто для описанных целей используют модель «сущность-связь», цель создания которой заключается в отображении сущностей моделируемой предметной области, атрибутов сущностей, описывающих характеристики объектов и имеющихся связей. Получаемая в ходе этого этапа графическая модель необходима для наглядного представления и последующего обсуждения основных концепций с пользователями будущей системы. При этом для графического представления применяют нотацию наиболее популярного сегодня графического унифицированного языка моделирования UML.

Метамодельное проектирование представляет собой своеобразное логическое проектирование, то есть описание элементов концептуальной модели в понятиях выбранной модели данных. Преобразование концептуальной модели в логическую модель должно осуществляться по определённым ранее формальным правилам. Т.е. при метамодельном проектировании учитывается специфика выбранной конкретной модели данных. При этом не учитывается специфика конкретной реализации структуры БД и используемой СУБД. Для данного вида проектирования используется унифицированная метамодель объектной системы [171, 209]. При этом для формализации представления моделей прикладных предметных областей в понятиях унифицированной метамодели

объектно-ориентированных приложений баз данных и выполнения её проверки (валидации и верификации) используется разработанный автором формальный математический аппарат [215-216]. Для упрощения графического представления результатов данного проектирования был разработан пользовательский UML-профиль [110].

Многие новые современные программные продукты разрабатываются с помощью объектно-ориентированных языков программирования (ООЯП). Такая популярность вызвана наличием апробированной многолетним использованием ОО-парадигмы и наличия инкапсуляции, наследования и полиморфизма, позволяющих повторно использовать написанные ранее не только программные классы, но и целые компоненты. Это сокращает время на разработку как нового продукта, так и на доработку существующего приложения. Таким образом, возникает задача представления модели предметной области в понятиях выбранного языка программирования. Именно для этого предназначено сущностное проектирование, суть которого заключается в выделении сущностей предметной области после метамодельного проектирования в виде синтаксических конструкций выбранного языка программирования. По своей сути это первая программная реализация приложения и она зависит от выбранного языка программирования. Наличие её позволяет прототипировать программный продукт еще на этапе проектирования. Кроме того, сущностное проектирование позволяет использовать такие подходы, которые не поддерживаются напрямую в языке программирования, так называемый «синтаксический сахар». Например, многие современные ООЯП не поддерживают механизм множественного наследования реализаций и технологию примесей, которая может быть представлена в виде синтаксической конструкции interface языка C#. Затем, описанные концепции могут быть реализованы с помощью дополнительного

программного кода, который генерируется программно (например, через CodeDOM) [59, 218].

Имплементационное проектирование необходимо для реализации модели, полученной при сущностном проектировании в виде синтаксических конструкций выбранного языка программирования и объектов баз данных. Т.к. в настоящее время чаще всего применяют ООЯП, а для сохранения информации реляционную модель данных, то при имплементационном проектировании необходимо выполнить два вида проектирования. Первым является имплементационное объектно-ориентированное проектирование, суть которого заключается в построении UML-диаграммы классов, содержащей классы и ассоциации для выделенных ранее сущностей. Вторым является имплементационное реляционное проектирование, цель которого заключается в создании ER-диаграммы реляционных отношений для выбранной СУБД.

Презентационное проектирование применяется для наглядного представления графических форм приложения, предназначенных для просмотра и редактирования существующей информации и ввода новой [168]. В ходе данного этапа создается набор моделей, содержащих графическое представление сущностей предметной области в разработанной нотации [160]. Т.е. разрабатывается макет графического интерфейса пользователя, содержащий расположения отдельных элементов. Выполнение данного этапа позволяет согласовать внешний вид приложения на ранней стадии выполнения проекта.

Несмотря на наличие множества трудоёмких, с точки зрения разработчика, этапов, часть из них можно автоматизировать, т.е. часть моделей можно генерировать автоматически на основе моделей, полученных в ходе выполнения другого этапа предложенной методологии. Опыт показал, что разработчику информационной системы необходимо корректно выполнить только метамодельное проектирование. Т.е. благодаря наличию

собственного UML-профиля нет необходимости в концептуальном проектировании, т.к. имеет смысл сразу проектировать в понятиях экземпляров метаклассов объектной системы. Благодаря использованию диаграмм классов языка UML, применяемых при метамодельном проектировании, созданные диаграммы просты для понимания конечными пользователями. Сущностное проектирование может быть выполнено автоматически на основании информации, вносимой в ходе метамодельного проектирования, добавив соответствующие атрибуты и набор предопределённых значений в полученную UML-диаграмму классов. Имплементационное проектирование также можно автоматизировать с помощью генераторов программного кода, позволяющих создавать имплементационные ОО-модели. Применением методов (паттернов) объектно-ориентированного отображения можно автоматизировать имплементационное реляционное проектирование для РСУБД. Для автоматизации презентационного проектирования необходимо отделить представление данных в интерфейсе пользователя от механизмов хранения данных. Предоставляя механизмы настройки графического интерфейса конечному пользователю, можно добиться быстрого прототипирования приложения и освободить разработчика от этой обязанности. Описанная методология успешно реализована в созданной автором унифицированной среде быстрой разработки корпоративных информационных систем SharpArchitect RAD Studio и протестирована на множестве различных проектов, в ходе которых реализовались приложения баз данных для прикладных предметных областей [226].

Для наглядной демонстрации всех этапов метамодельного проектирования и результирующих моделей необходимо выбрать любую прикладную предметную область. Рассмотрим в качестве примера

унифицированную модель тестирования инструментов разработки объектно-ориентированных приложений [164, 211].

Перед проектированием унифицированной модели тестирования были выдвинуты критерии оптимальности (КО), представляющие собой требования о наличии определенных структурных элементов на диаграмме классов и которым должна соответствовать готовая реализация. Были выдвинуты следующие требования для унифицированной модели тестирования инструментов разработки объектно-ориентированных приложений:

1. Необходимо наличие глубоких иерархий наследования. В реальных приложениях очень часто появляются глубокие иерархии, представляемые отношением наследования и объединяющие транзитивно не менее трех классов.
2. Присутствие нескольких иерархий наследования. Это позволит продемонстрировать различные опции и режимы, имеющиеся в инструменте.
3. Наличие абстрактных классов в иерархии. Абстрактные классы не могут иметь экземпляров в системе и описаны в качестве контейнеров для атрибутов и методов, используемых в производных реальных (инстанцируемых) классах.
4. Присутствие множественных (n-арных) ассоциаций. В приложениях, автоматизирующих реальные прикладные области, часто возникают ассоциации, в которых участвуют три и более классов. Подобные отношения называют множественными или n-арными ассоциациями.
5. Наличие ассоциаций с атрибутами. Многие предметные области содержат атрибуты, которые не принадлежат определенным сущностям, и их значения появляются только при организации связей между экземплярами классов. Поэтому проектируемая

унифицированная модель должна иметь ассоциации с атрибутами.

6. Присутствие композиции между классами. Композиция - это ассоциация между классами, представляющими Часть и Целое. Особенность в том, что класс, представляющий Часть, может входить только в один экземпляр класса, представляющий Целое. При этом класс, представляющий Целое, управляет жизненным циклом класса, представляющего Часть. Т.е. при удалении Целого, Части также удаляются. Эта особенность поведения очень важна для многих прикладных предметных областей.
7. Наличие рекурсивных ассоциаций. Рекурсивными называют ассоциации, концы которых связывают один и тот же класс. Подобные отношения позволяют реализовать иерархию подчинения.
8. Наличие ассоциаций между классами, входящими в одну иерархию наследования. С точки зрения реализации необходимо предусмотреть ассоциации, края которых связывают классы, входящие в одну иерархию наследования, т.е. противопоставляют базовый и дочерний класс друг другу.
9. Присутствие класса-ассоциации. Класс-ассоциация - это ассоциация, которая в то же время является и классом. Особенность использования в том, что класс-ассоциация представляет собой уникальную ассоциацию, т.е. комбинация экземпляров классов в этой ассоциации является уникальной.
10. Наличие ассоциаций между классом-ассоциацией и другим классом. С теоретической точки зрения, класс-ассоциация - это класс, поэтому он может участвовать в других ассоциациях. С точки зрения реализации, класс-ассоциация представляет собой класс, содержащий атрибуты (поля или свойства языка программирования), которые ссылаются на другие классы. В свою очередь для организации ассоциации с классом-ассоциацией

необходимо в зависимом классе создать атрибут, типом которого выступает класс-ассоциация.

11. Присутствие в модели перечислений. С теоретической точки зрения перечисление - это набор предопределенных констант, при этом пользователю нельзя расширить этот набор путем добавления новых значений.

Метамодель не может существовать отдельно от использующего его инструмента разработки приложений. Описываемая унифицированная метамодель объектной системы используется в инструменте быстрой разработки корпоративных приложений SharpArchitect RAD Studio. Создание среды разработки начнем с формирования требований, критериев оптимальности (КО), которым должна удовлетворять реализация среды разработки (КОср). Т.к. рассматриваемая среда предназначена для разработки корпоративных информационных систем, то созданные КИС также должны удовлетворять требованиям, предъявляемым к КИС [46, 177, 193], на основании которых сформулированы следующие КОср для SharpArchitect RAD Studio [217, 224, 226]:

1. Предусмотреть возможность описания любых прикладных предметных областей. Заранее невозможно предугадать сферу применения системы, поэтому необходим общий способ описания прикладных предметных областей. Имеет смысл использовать структурные диаграммы (например, диаграммы классов) языка UML и на основе их интерпретации строить масштабируемые КИС [46, 193].
2. Реализовать поддержку различных архитектур приложений. В настоящее время существует множество различных архитектур от классической двухзвенной "файл-сервер" до многозвенной архитектуры с поддержкой "распределенных объектов" [35, 45]. Выбор оптимальной структуры во многом зависит от масштаба реализуемой КИС и аппаратных возможностей, имеющихся у предприятия. Выполнение данного КО позволит своевременно и с минимальными затратами масштабировать приложения при возрастающих потребностях организации.

3. Использование современного языка программирования и современной СУБД. Популярной парадигмой сегодня является объектно-ориентированное программирование, поэтому использование соответствующего языка программирования является наиболее целесообразным [35]. Для хранения информации в настоящее время чаще всего используют реляционные системы управления базами данных (РСУБД). Применение современных технологий позволяет увеличить срок эксплуатации системы и сократить затраты на ее владение [7].

4. Реализовать инфраструктуру безопасности. Разграничение прав доступа к данным и отображение минимального набора информации, достаточного лишь для выполнения должностных обязанностей - являются ключевыми особенностями разработки современных КИС [46]. Имеет смысл предусмотреть возможность двух способов проверки подлинности пользователей: 1) внутренней - основанной на учетных записях, имеющихся в КИС (чаще всего используется для Веб-приложений) и 2) встроенной - основанной на механизмах, предоставляемых операционной системой.

5. Поддержка механизмов расширения функциональных возможностей среды разработки. Современные языки программирования предоставляют обширную метаинформацию [35], что позволяет реализовать плагинную архитектуру, поддерживающую установку модулей расширений, представляющих собой dll-библиотеки [217]. Этот подход разрешает безгранично увеличивать функционал системы и её гибкость.

Для описания математической модели, позволяющей представить объектную модель информационной системы любой прикладной предметной области, будем использовать формальный аппарат теории множеств. Дальнейшим развитием данного решения может быть модельно-ориентированный подход, при котором понятие типа основано на теоретико-множественной модели, то есть с помощью описания x: T, "x имеет тип T", что эквивалентно принадлежности (x ∈ T,"x является членом множества T").

При этом все операции программы можно моделировать как набор манипуляций с множествами.

При проектировании информационных системы необходимо построить модель предметной области, содержащей различные типы сущностей.

В настоящий момент наибольшее количество новых приложений разрабатывается с применением объектно-ориентированного подхода. Эта парадигма, основанная на технологии наследования, позволяет многократно использовать разработанные ранее элементы, реализованные в виде классов. В результате сокращается время и затраты на разработку всей информационной системы, что является ключевым преимуществом при создании крупных программных продуктов. Подобные системы, как правило, являются многопользовательскими. При этом каждой категории пользователя необходима лишь часть имеющейся информации. К остальной информации необходимо запретить доступ пользователя. Т.е. возникает задача разграничения прав для многопользовательских приложений. В этом разделе представлена модель разграничения прав доступа для объектно-ориентированных приложений, созданная автором и использованная многократно при разработке крупных программных приложений.

Стандартным графическим языком моделирования различных аспектов объектных систем является язык UML. Этот язык, а именно структурные диаграммы классов, будут рассмотрены далее. В итоге под унифицированной моделью тестирования инструментов разработки объектно-ориентированных приложений будем понимать диаграмму классов, состоящую из классов и атрибутов и содержащих наиболее часто встречающиеся на практике отношения классов.

Немного о системах массового обслуживания

Системы обслуживания клиентов используются повсеместно, поэтому интерес к их изучению, формализации и автоматизации предметных областей возник довольно давно [49, 51]. С точки зрения разработки информационных систем, предметная область может быть представлена в виде рис. 5.7. Через информационную систему (IS) последовательно регистрируются заказы (O) клиентов (C). Каждый сотрудник (W) последовательно получает свой заказ и исполняет параллельно на исполнительном устройстве (E). Выполненный заказ (EO) последовательно регистрируется работником и выдается клиенту.

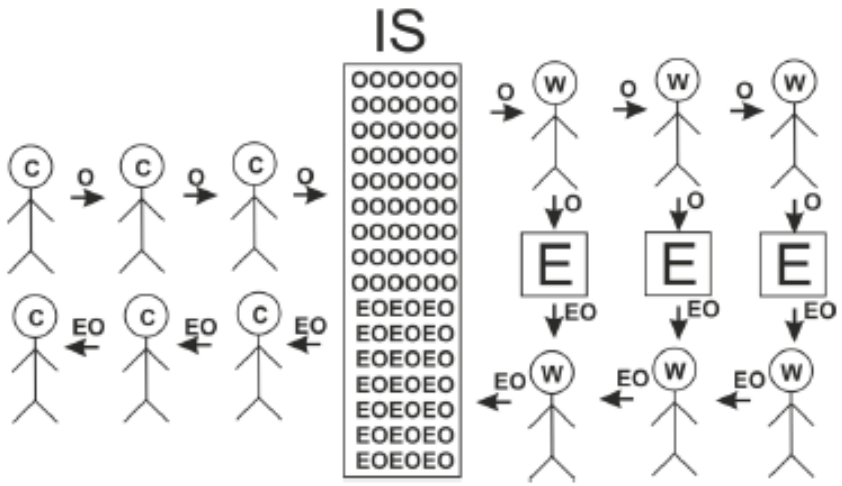

Рис. 5.7. Использование информационных систем в предметной области массового обслуживания клиентов

Рассмотрим имеющиеся работы. Так в работе [25], опубликованной в 1991 году, автор обосновывает необходимость создания программного комплекса, позволяющего имитировать поведение посетителей ресторанов и создающего различные типовые операции бизнес-процессов. Описан состав и структура модулей ИС, а также в виде диаграммы представлен поток выполнения различных тестов. В конце работы автор описывает возникающие проблемы валидации и верификации созданных моделей.

В [72] авторы рассматривают реализованную в корпорации Toshiba информационную систему, которая распознает устную речь клиента и на основе выделенных корней формирует запросы на создание заказов в системе обслуживания клиентов. Работа написана в 1992 году и подробно раскрывает процесс выделения ключевых слов, определения корня слов, рассматривает пример распознания голоса и последующего распарсивания слов и последовательностей. Затем на основе имеющейся грамматики находит соответствие с продуктами в ИС ресторана и формируется заказ на приготовление.

Более поздняя работа [78], написанная в 2008 году, представляет собственный метод оценки удовлетворенности клиентов услугами ресторанов быстрого обслуживания, который авторы назвали TOPSIS. Этот подход основан на многофакторном анализе, при котором все выделенные критерии были распределены на несколько групп. Авторы предлагают собственный формальный аппарат и набор формул, которые позволяют ранжировать каждый критерий в пределах группы на основе рассчитанного веса. Затем веса суммируются и получается интегрированный показатель. Предложенный подход апробирован в ресторанах США и Китая.

В 2009 году в [68] подробно описана реализация многозвенной информационной системы, позволяющей полностью автоматизировать систему массового обслуживания и все бизнес-процессы от формирования заказа клиентом до получения заказа. В начале работы авторы рассматривают процесс формирования заказов с точки зрения различных участников, таких как клиент и обслуживающая система. В результате найденных отличий делается вывод о необходимости реализации различных звеньев системы и различных интерфейсов.

В том же 2009 году в работе [79] авторы рассмотрели процесс разработки программного продукта, который использует карманные персональные компьютеры (КПК, PDA) для создания и обслуживания заказов клиентов. Представлен сенсорно-ориентированный графический интерфейс и описаны основные задачи, выполняемые с его помощь.

Проблема внедрения современных сенсорно-ориентированных приложений рассмотрена и в работе [10], опубликованной в 2010 году и посвященной процессу автоматизации системы обслуживания клиентов на примере столовой. Авторы начали с анализа бизнес-процессов и выделения ключевых процедур, которые в конечном итоге были распределены на несколько групп, каждая из которых выполняется либо с помощью сенсорно-ориентированного приложения, либо с помощью приложения, использующего манипулятор мышь. С аппаратной точки зрения было предложено сделать специальные виды обеденных столов, схема которых

представлена на рисунке в той статье и позволяет понять расположение системного блока, сенсорного экрана и проектора.

В том же 2010 году в [8] авторы выполнили реинжиниринг процесса приготовления комплексных обедов, а затем на основании построенных диаграмм осуществили оптимизацию процесса. Это позволило построить диаграмму «сущность-связь» в понятиях реляционной модели данных.

Одной из последних работ, опубликованных в 2014 году и посвященных разработке ИС для ресторанов, является работа [20]. Авторы описывают собственный продукт, который позволяет заказывать товары самостоятельно с помощью приложений, запущенных на сенсорных устройствах. При этом в начале (только один раз) необходимо ввести дополнительную информацию о себе, которая потом используется при формировании заказов и сокращает тем самым время обслуживания.

В 2014 году также была опубликована работа [33], которая посвящена процессу моделирования системы массового обслуживания для ресторанов быстрого питания. В ней построена модель очереди обслуживания клиентов типового реально существующего кафе. В качестве математического аппарата использовалась модель марковских процессов.

Для программной инженерии

Перед реализацией любого программного продукта необходимо выделить критерии оптимальности (КО), которые содержат функциональные особенности будущей реализации. Сами критерии были выдвинуты после изучения функционала, имеющегося у программного продукта iiko, а также иных продуктов, построенных на базе 1С. Для информационных систем, автоматизирующих предметные области обслуживания клиентов, выделены следующие требования, определяющие необходимость:

1. Предусмотреть возможность автоматизации с помощью единой системы как небольшое заведение, так и целой сети заведений ($КО_1$);
2. Разработать развитый графический интерфейс с поддержкой сенсорных экранов ($КО_2$);
3. Реализовать многопользовательский учет заказов, поступающих от клиентов ($КО_3$);
4. Реализовать гибкую архитектуру приложения с возможностью расширения в будущем ($КО_4$);
5. Предусмотреть возможность интеграции с различными периферийными устройствами ($КО_5$).

Рассмотрим каждый критерий более подробно. Требования $КО_1$ предполагают использование единой ИС независимо от масштабов организации: от одиночного ресторана до крупной сети. Часто подобные требования называют масштабируемостью системы. Ключевым достоинством в свете рассматриваемой задачи является снижение затрат на обучение персонала при естественном расширении организации. Бизнесу свойственно расширяться, и со временем одно-единственное заведение превращается в крупную сеть обслуживания клиентов. В конечном итоге этот подход приведет к повышению прибыли от экономической деятельности.

Применение сенсорных экранов позволяет исключить клавиатуру и мышь и управлять пальцами рук. Это значительно сокращает время на выполнение типовых бизнес-процессов, что очень важно в системах массового обслуживания. Именно поэтому программе необходим развитый графический интерфейс с поддержкой сенсорных экранов, что описано в требованиях $КО_2$. Анализ функциональных возможностей самой популярной в России системы iiko, а также наблюдение за устройствами, применяемыми в различных предметных областях систем массового обслуживания, показал, что разрешение используемых сенсорных экранов составляет 1024×768 пикселей, и на нем необходимо оптимально разместить выводимую информацию и элементы графического интерфейса пользователя.

Бизнес-процессы систем обслуживания клиентов в упрощенном виде предполагают наличие одного компьютера с установленным программным обеспечением, доступ к которому имеет множество пользователей системы, оформляющих и обслуживающих заявки клиентов. Так как к одному компьютеру имеют доступ несколько сотрудников, то необходим механизм идентификации и разграничения прав доступа. Для этого достаточно каждому пользователю системы присвоить уникальный код, который может быть введен любым удобным способом. Чтобы сократить время на ввод кода, часто используются прокси-карты и их считыватели. Исходя из этого, каждый пользователь, подходя к компьютеру, подносит личную прокси-карту к считывающему элементу и, авторизовавшись в системе, выполняет требуемые действия. Затем он выходит из системы, нажав кнопку на графической форме, и подходит следующий пользователь. При этом предыдущий пользователь может приступить в этот момент к выполнению заказа клиента. Следовательно, скорость обслуживания во многом зависит от скорости входа в систему, и использование прокси-карт является оптимальным решением в данном случае. Таким образом, можно реализовать оперативный многопользовательский учет заказами, что записано выше как $КО_3$. При этом различным группам пользователей необходимо предоставить различный интерфейс, оптимальный для выполнения их должностных обязанностей. Например, одной категории сотрудников необходимо вносить и редактировать заявки клиентов. Менеджеру необходимо лишь просматривать и иметь возможность удалять ошибочные заказы из системы.

$КО_4$ требует реализовать гибкую архитектуру приложения с возможностью расширения в будущем. Прогресс не стоит на месте, постоянно появляются новые устройства и технологии. Возникают все новые задачи, например, задача интеграции разработанной системы с различными бухгалтерскими программами. В этой связи разработка гибкой архитектуры позволит сохранить инвестиции в будущем, когда потребуются серьезные доработки системы. В настоящее время для разработки новых программных продуктов используются чаще всего объектно-ориентированные (ОО) языки программирования, основными свойствами которых являются инкапсуляция, полиморфизм, наследование. При реализации программного обеспечения применяют шаблоны (паттерны) проектирования, которые представляют собой повторно используемые архитектурные решения. Для сохранения информации в долговременной памяти используется база данных (БД), управляемая системой управления базами данных (СУБД). В настоящее время наиболее популярными являются реляционные СУБД. Существует множество различных библиотек, позволяющих реализовать взаимодействие любых реляционных СУБД для организации взаимодействия с другими системами. Из-за наличия существенных различий в организации и обработке данных в объектно-ориентированных языках программирования (ООЯП) и в реляционных системах управления данными возникает объектно-реляционное несоответствие, для преодоления последствий которого используют методы (шаблоны, паттерны) объектно-реляционного отображения. Популярные подходы решения описанной проблемы подробно представлены в работах [45, 48, 171-172, 226-227]. Таким образом, для соответствия $КО_4$ одним из решений является разработка клиентского приложения на ООЯП, а в качестве хранилища информации необходимо выбрать реляционную СУБД.

Возможность интеграции с различными периферийными устройствами особенно актуальна в нашем случае, так как к рабочему месту пользователя ИС подключены такие устройства, как принтер чеков (подключенный по USB или RS-232), использующий для вывода на печать рулон бумаги шириной 8 см, так и считыватели прокси-карт. Разнообразие периферийных устройств постоянно растет, что связано с уменьшением скорости печати заданий и тем самым приводит к сокращению времени обслуживания клиентов. Описанное выше позволило сформировать требования $КО_5$.

Разработка любого программного обеспечения начинается с формулирования задач, которые необходимо решить с помощью него. Поэтому необходимо сформировать ряд требований (критерия оптимальности), которым будет соответствовать готовая реализация:

1. Система должна получать данные от различных библиотек, предоставляемых брокерами или биржами. Так как каждая биржа и многие брокеры предоставляют библиотеки, позволяющие получать данные, то недопустимо иметь жесткую привязку к каким-либо внешним интерфейсам или классам. Вместо этого потребуется написать ряд вспомогательных классов системы, позволяющих реализовать унифицированный подход при подключении сторонних библиотек.

2. Активное использование многопоточности. Современные процессоры являются многоядерными и поддерживают ряд технологий, позволяющих обеспечить параллельное выполнение нескольких фрагментов кода одновременно. Именно поэтому имеет смысл при разработке платформы предусмотреть выделение отдельных потоков, в рамках которых будут выполняться такие операции, как загрузка данных, получаемых от биржи/брокера, а также выполнение различных шагов биржевых роботов.

3. Необходимо реализовать развитый графический интерфейс пользователя. Наличие графический форм, отображающих информацию в удобном для пользователе виде – основное требование, предъявляемое к любому биржевому программному обеспечению. Особенно это актуально в отношении описываемой системы, ведь трейдер должен отслеживать текущую ситуацию и процесс функционирования робота и, в случае возникновения непредвиденных ситуаций (как правило, на ранних стадиях отладки алгоритма), вовремя остановить торговлю.
4. Возможность реализации множества различных биржевых роботов в рамках единой платформы. Это подразумевает наличие единой иерархии базовых классов, которые используются при создании роботов. Т.е. классы, реализующие финансовые стратегии биржевых роботов, будут унаследованы от имеющихся базовых классов.

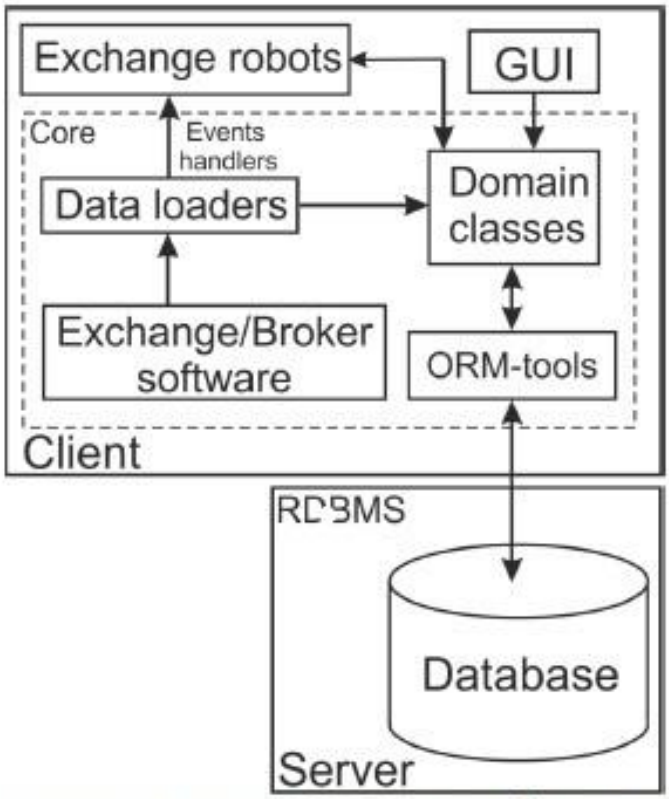

Рис. Структурная схема реализации платформы для создания и управления торговыми биржевыми роботами

Разработанная система тесно интегрируется с программным обеспечением, поставляемым биржей/брокером. Для этого используется иерархия классов, позволяющая выполнять загрузку данных из различных потоков (котировки, заявки, сделки и т.п.) в объекты классов предметной области. Таким образом реализуется связь между различными объектами и соответствующими коллекциями (например, связь между финансовым инструментом и заявками на него).

Биржевые роботы в рассматриваемой реализации представляют собой объекты (экземпляры классов), которые подписываются на обработчики событий, объявленные в классах загрузки данных. Т.е. каждый шаг финансовой стратегии выполняется после поступления новых данных от программных библиотек биржи/брокера. В результате происходит создание объектов предметной области и посылка команд в библиотеки биржевого программного обеспечения для создания/перемещения/отмены заявок на покупку/продажу контрактов торгового инструмента.

При разработке активно используется многопоточность. Такие операции, как получение данных от библиотек брокера и выполнение шагов финансовой стратегии, составляющих основу алгоритма робота, выполняются в отдельном дочернем потоке. В основном потоке выполняется лишь обновление графического интерфейса пользователя, что позволяет трейдеру контролировать процесс торговли. Эта особенность связана с принципами реализации графических библиотек в операционной системе Windows в общем и в платформе .Net Framework в частности [242].

Лекция № 3 составлена на основе материалов диссертации « Методология и инструментарий метамодельного проектирования приложений баз данных»

Автор Олейник П.П.

(если Вам в Вашем диссертационном исследовании нужна тематика этой работы – обращайтесь bashlykova_a_a_mirea@mail.ru за получением полного текста работы)