МОДЕЛИРОВАНИЕ ПРИЛОЖЕНИЯ

Постановка задачи

Разработать систему информационной поддержки процессов, связанных с приготовлением пищи. Система разрабатывается для персонального компьютера и призвана освободить человека от необходимости ручного ведения каталога (справочника, книги) рецептов и ручного планирования питания на определенный период. Система должна предусматривать выполнение всех операций, необходимых для ведения полнофункциональной базы данных рецептов (добавление, модификация, просмотр), а также использование базы для составления меню и планирования питания. Система должна работать в диалоговом режиме, быть удобной и интуитивно понятной для использования.

План работы

Перед тем как приступить к работе по кодированию программы, которая бы решала поставленную задачу, попробуем смоделировать эту систему. В процессе моделирования мы будем вынуждены пройти следующие этапы:

1. Этап ***анализа***. Состоит из следующих подэтапов:
   - **определение вариантов использования**, сформулированных в постановке задачи;
   - **выделение основных (важных для нашей задачи) объектов предметной области** и основных компонент, возможно, искусственных с точки зрения предметной области, но естественных для решения нашей задачи;
   - **формулировка обязанностей компонент** перед другими компонентами и пользователем (ответственности компонент);
   - **описание сценариев взаимодействия системы и пользователя** в соответствии с определенными вариантами использования, а также внутренних (межкомпонентных) механизмов достижения поставленных целей.
2. Этап ***проектирования***. Этап подразумевает формализацию идей и решений, сформулированных в процессе анализа. На этом этапе:
   - определяются **архитектурные** составляющие системы, т. е. классы;
   - описывается **взаимодействие классов** между собой с учетом их ответственности;
   - **уточняются сценарии** работы в проекции на определенные классы.
3. Этап ***генерации кода***. В зависимости от выбранного языка программирования необходимы языкозависимые уточнения модели.

Анализ и концептуальная модель

Для анализа проектируемой системы и построения концептуальной модели будем использовать диаграммы вариантов использования.

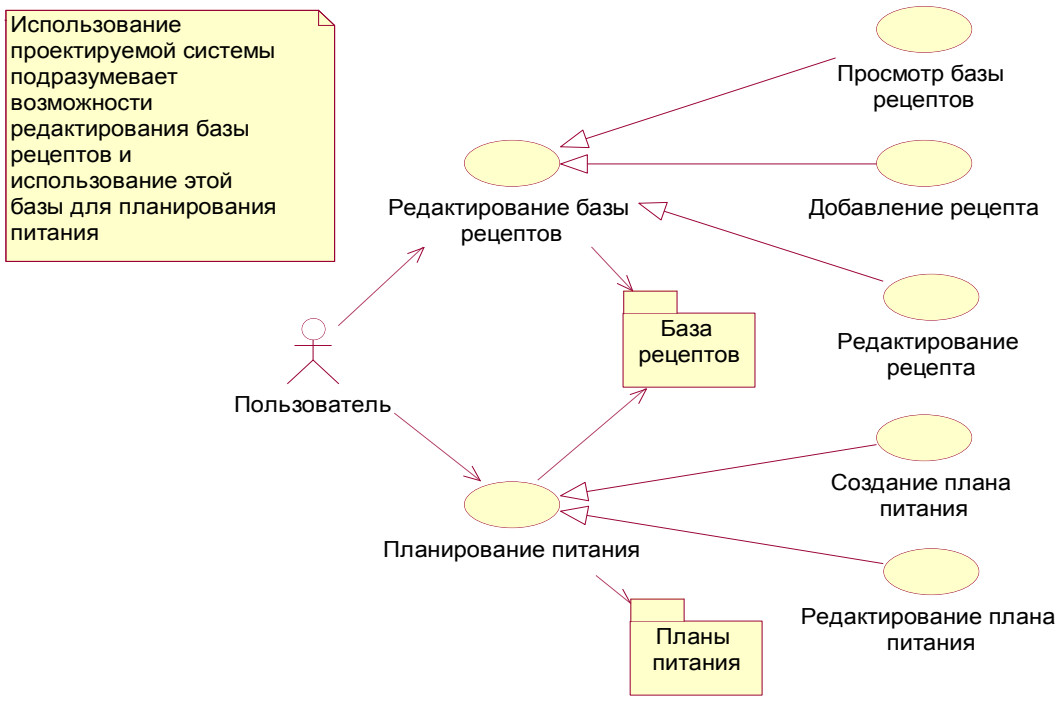

Рис. 6.32. Диаграмма вариантов использования

Разрабатываемая система подразумевает наличие заинтересованного в ней человека, который будет работать с системой, задействовав ее возможности в своей повседневной жизни. В связи с этим необходимо на этапе анализа рассмотрение действующего *активного внешнего объекта* – **Пользователя**.

Система разрабатывается для *ведения базы данных* рецептов (далее – база рецептов) и для *планирования питания*.

Диаграмма на рис. 6.32 отображает эти возможности более детально. *Редактирование* базы рецептов **детализировано** тремя различными видами деятельности – *просмотр базы*, *добавление* и *редактирование рецептов*. *Планирование питания* обобщает два вида действий – *создание нового плана питания* и *редактирование* уже существующего.

Кроме того, любое редактирование базы рецептов подразумевает использование набора (категории) классов, непосредственно связанных с базой рецептов, а планирование питания в дополнение должно использовать еще и набор классов, связанных с планами питания. Рассмотрим эти *категории классов* и связанные с ними *варианты использования* отдельно (рис. 6.33 и 6.34).

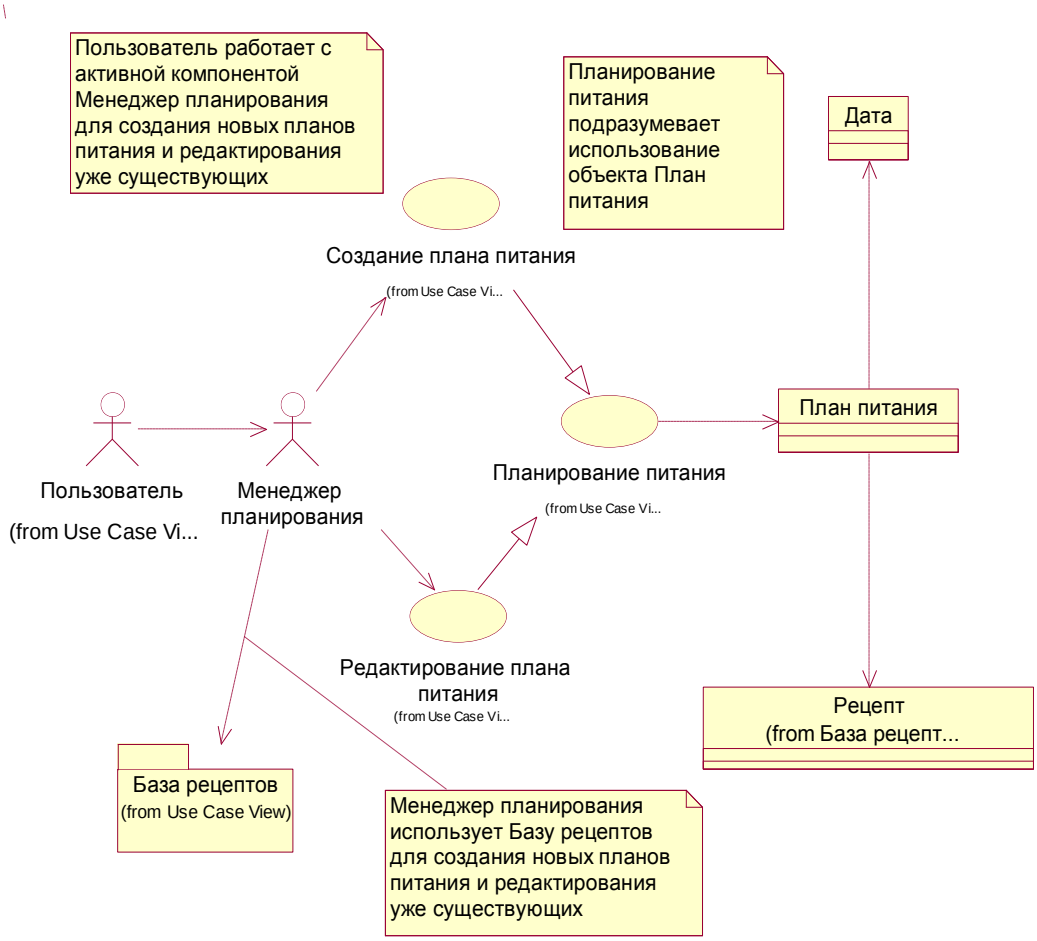

Рис. 6.33. Варианты использования для планирования питания

Для организации работы с планами питания необходимо введение *активного* компонент "**Менеджер планирования**" Активным он является потому, что может на основании внешнего запроса от Пользователя непосредственно осуществлять работу над планами питания. Пользователь же *использует* Менеджер планирования для любых действий, связанных с планированием питания. Менеджер планирования "умеет" на основании данных ему инструкций *создавать планы* питания и *редактировать* уже существующие. В своей деятельности Менеджер планирования использует категорию классов, работающую с базой рецептов.

Естественным и логичным шагом будет внесение на диаграмму класса **План питания**, с экземплярами которого и будет происходить работа. Это в свою очередь добавляет две связанные с **Планом питания** сущности – **Дата** и **Рецепт.**

Фактически **План питания** будет представлять собой набор (список) рецептов с указанием планируемой даты его использования.

Перейдем к рассмотрению категории вариантов использования, связанных с базой рецептов (рис. 6.34). По аналогии с Менеджером планирования введем *активный объект* **Менеджер рецептов**, который будет обрабатывать все запросы к базе рецептов, тем самым, централизуя их вызовы. Его появление на диаграмме тем более обусловлено фактическим наличием двух

активных пользователей базы рецептов – **Менеджера планирования** и **Пользователя**. Для удобства обозначим таких пользователей обобщающим понятием "**Пользователь базы рецептов**".

Менеджер рецептов полнофункционально обеспечивает работу с базой рецептов, а именно по соответствующим запросам *добавляет* новые рецепты, *редактирует* уже существующие, *обеспечивает просмотр* базы рецептов. Естественно ввести на эту диаграмму класс **Рецепт**, с экземплярами которого и будет происходить работа по модификации и просмотру базы рецептов.

Легко видеть, что в процессе рассуждений была сформирована **концептуальная модель** предметной области нашего Кухонного Помощника (рис. 6.35).

Так или иначе, доступ ко всей информации, с которой работает Кухонный Помощник (далее – КП), будет происходить через *пользовательский интерфейс*, который, в свою очередь, использует активные компоненты **Менеджер планирования** и **Менеджер рецептов** для обработки поступающих запросов.

Менеджер рецептов будет работать с сущностями **Рецепт** и **База рецептов** (является хранилищем рецептов, средством их каталогизации и поиска).

Менеджер планирования работает с **Планами питания**, а необходимую для их модификации информацию по рецептам получает от **Менеджера рецептов.**

Также можно более четко сформулировать ответственность каждого объекта и каждой компоненты.

Пользовательский интерфейс должен:

1. Приветствовать пользователя в начале работы.
2. Обеспечивать возможность визуально:
   - получать доступ к функциональности Менеджера рецептов (просмотр базы, добавление и редактирование рецепта);
   - получать доступ к функциональности Менеджера планирования (создание нового плана питания и пересмотр существующего плана).
3. Вносить интерактивность в процесс работы – организация подтверждающих диалогов, подсказок по использованию и прочее.

Рецепт должен:

1. Содержать и предоставлять по запросу список ингредиентов.
2. Содержать и предоставлять по запросу список последовательных действий по приготовлению.
3. Менять по запросу ингредиенты.
4. Менять по запросу список действий.

Дата должна:

1. Содержать свое описание (день, месяц, год).
2. Содержать (и предоставлять по запросу) текстовые комментарии (праздник, день рождения).

План питания должен:

1. Содержать (и предоставлять по запросу) список последовательных дат и соответствующий им список блюд (рецептов).
2. Модифицировать расписание.

База рецептов должна:

1. Обеспечивать структурированный доступ к своему содержимому.
2. Предоставлять возможности по добавлению и модификации рецептов.

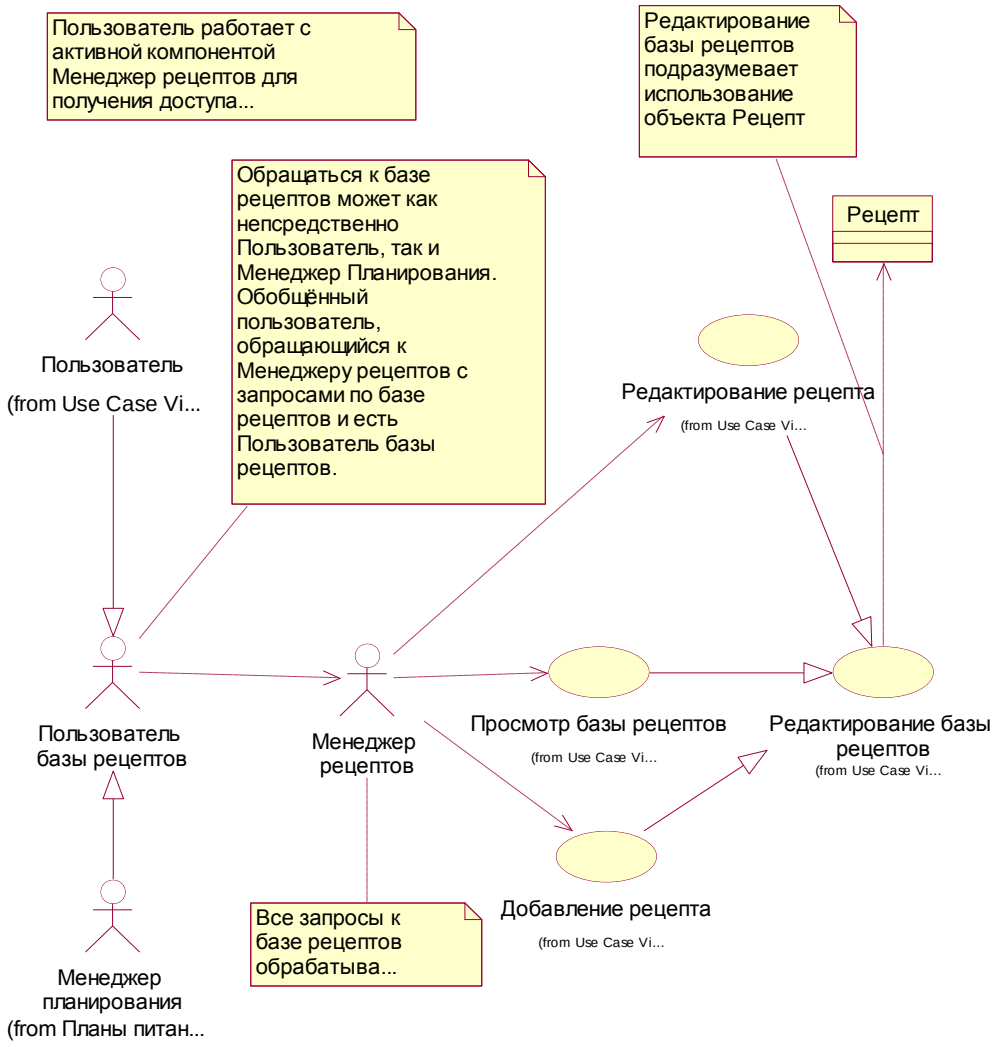

Рис. 6.34. Варианты использования для работы с базой рецептов

Менеджер рецептов должен:

1. Искать запрашиваемые рецепты в базе рецептов.
2. Создавать новые рецепты по введенным данным (определяя, что нужно спрашивать и в каком порядке).
3. Модифицировать рецепты с проверкой модификации на корректность.
4. Сохранять модификации в базе рецептов.

Менеджер планирования должен:

1. Уметь создать новый план питания (определяя, что нужно спрашивать и в каком порядке).
2. Редактировать план питания (проверяя корректность модификаций).
3. Уметь сохранять план.
4. Открывать и просматривать существующий план.
5. Посылать требуемый план питания на печать.

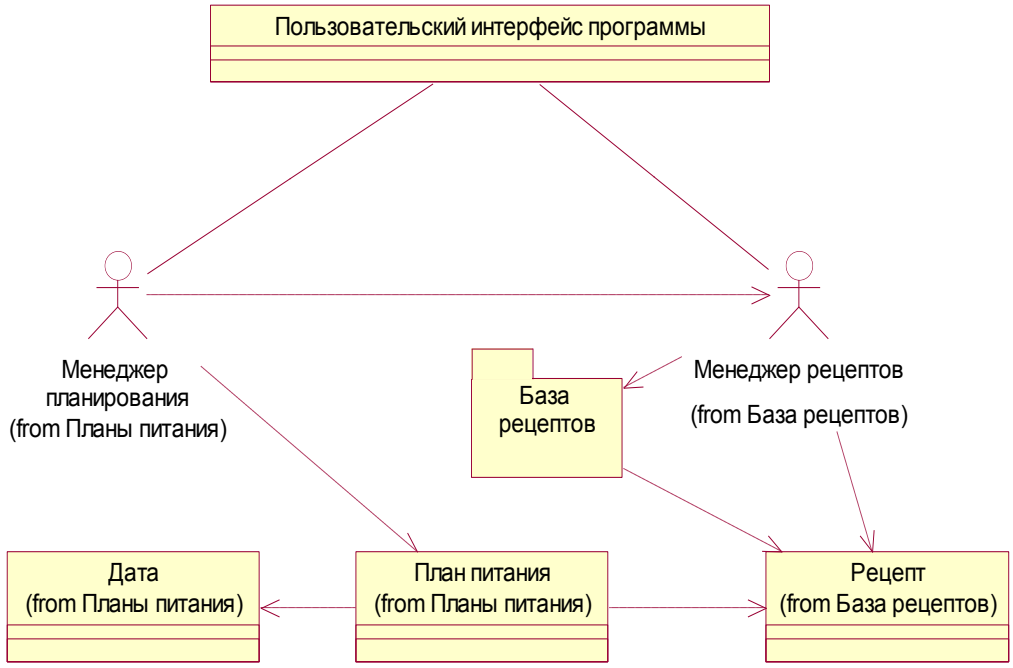

Рис. 6.35. Концептуальная модель

Сценарии работы

Для каждого варианта использования возможно составление сценариев работы. Возьмем для описания *основные потоки* каждого из рассмотренных вариантов использования и создадим для них диаграммы последовательности (Sequence diagrams).

Начнем с *добавления нового рецепта* в базу (рис. 6.36). Кроме естественно-необходимых действий, отображенных на диаграмме, следует особо отметить, что Менеджер рецептов сам инициирует получение параметров нового рецепта (шлет запрос Пользовательскому интерфейсу на отображение соответствующей формы). Также не совсем очевидным является действие, направленное на проверку целостности рецепта, осуществляемое также менеджером рецептов.

В качестве *другой возможности* отображения ответственности компонент и взаимодействия ее с другими компонентами уместно привести пример *диаграммы взаимодействий* (Collaboration diagrams), сгенерированной на основе приведенной выше диаграммы последовательности. Диаграмма приведена на рис. 6.37.

Однако для решения поставленной задачи более удобным и практичным средством моделирования являются диаграммы последовательности, поэтому в дальнейшей работе мы будем употреблять только их.

Диаграмма на рис. 6.38 отображает *единичный сеанс просмотра базы* рецептов, т. е. извлечение и просмотр одного рецепта, возможно, с последующим редактированием.

Из особых решений, принятых для взаимодействия объектов, следует отметить *организацию поиска* необходимого рецепта средствами базы рецептов.

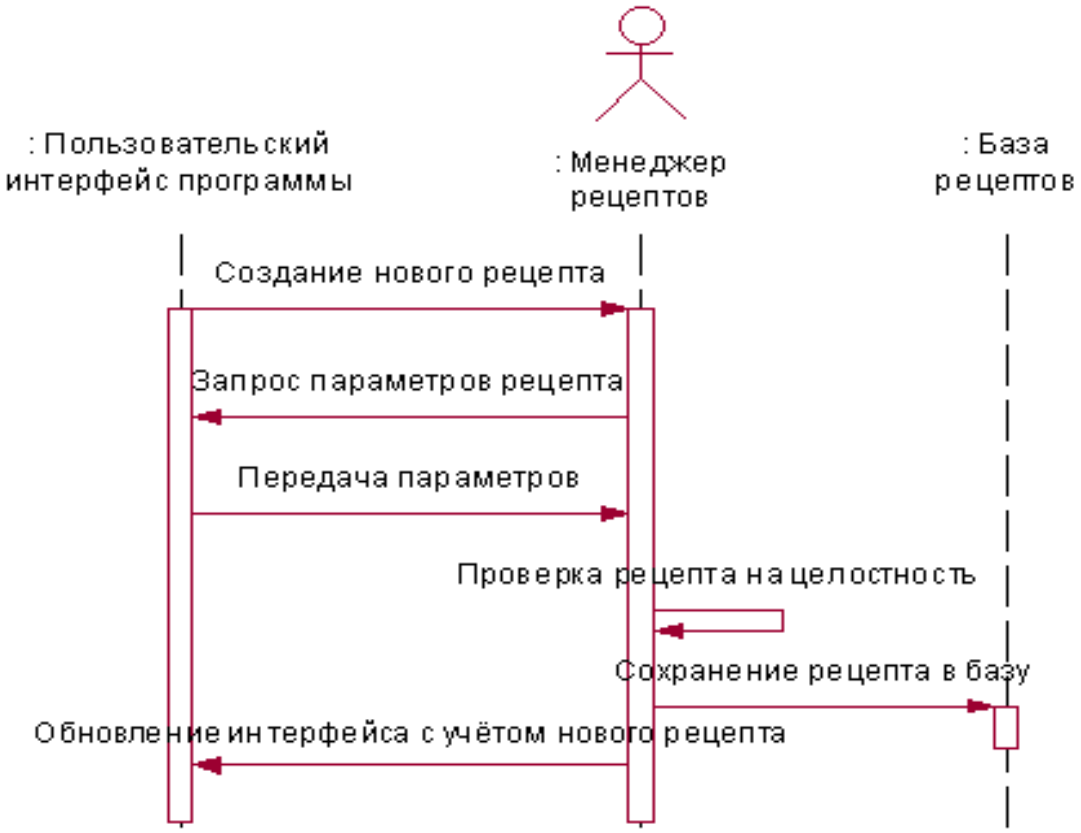

Рис. 6.36. Добавление нового рецепта

-

Рис. 6.37. Добавление нового рецепта

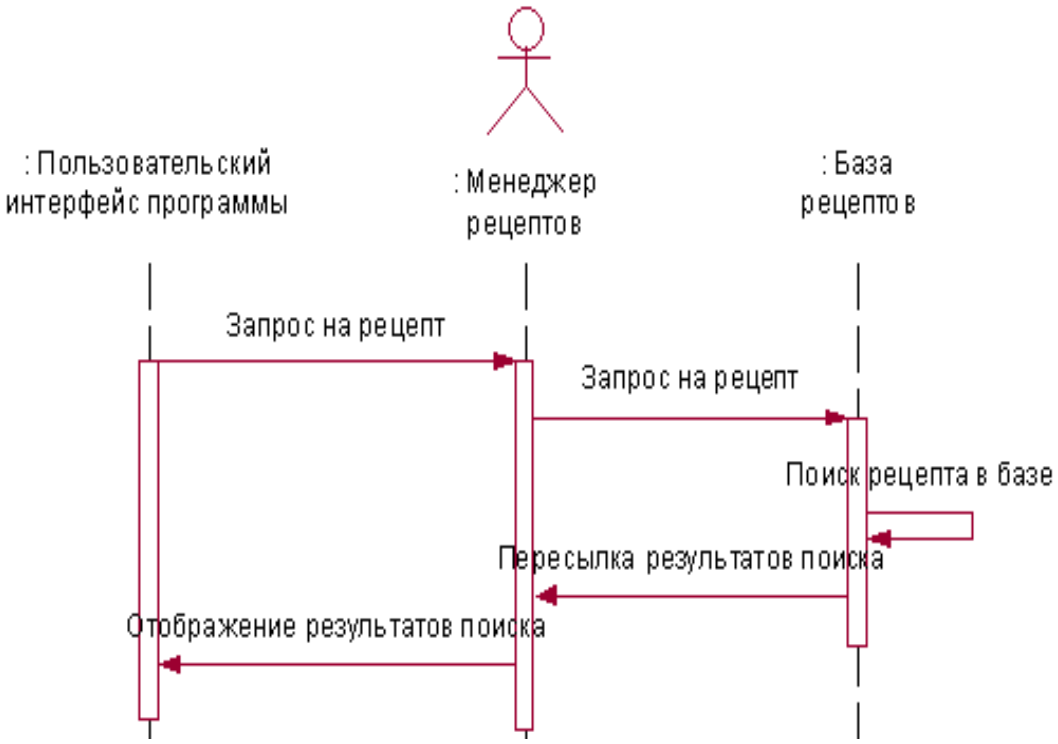

Рис. 6.38. Извлечение и просмотр рецепта

Диаграмма, приведенная на рис. 6.39, похожа на диаграмму добавления нового рецепта в базу и представляет *модификацию уже существующего рецепта*.

Следует отметить, что Менеджер рецептов перед добавлением рецепта в базу должен проверять, не нарушена ли его целостность. Так как мы описываем только *основной* поток, то предполагаем, что целостность не нарушена. После добавления рецепта в базу необходимо *обновить интерфейс* программы.

Теперь рассмотрим *основной* поток вариантов использования, связанных с планированием питания. На рис. 6.40 показана диаграмма последовательности *создания нового плана питания*. Рис. 6.41 отображает сценарий *модификации текущего плана*, а рис. 6.42 – *просмотр* (использование) *текущего плана*.

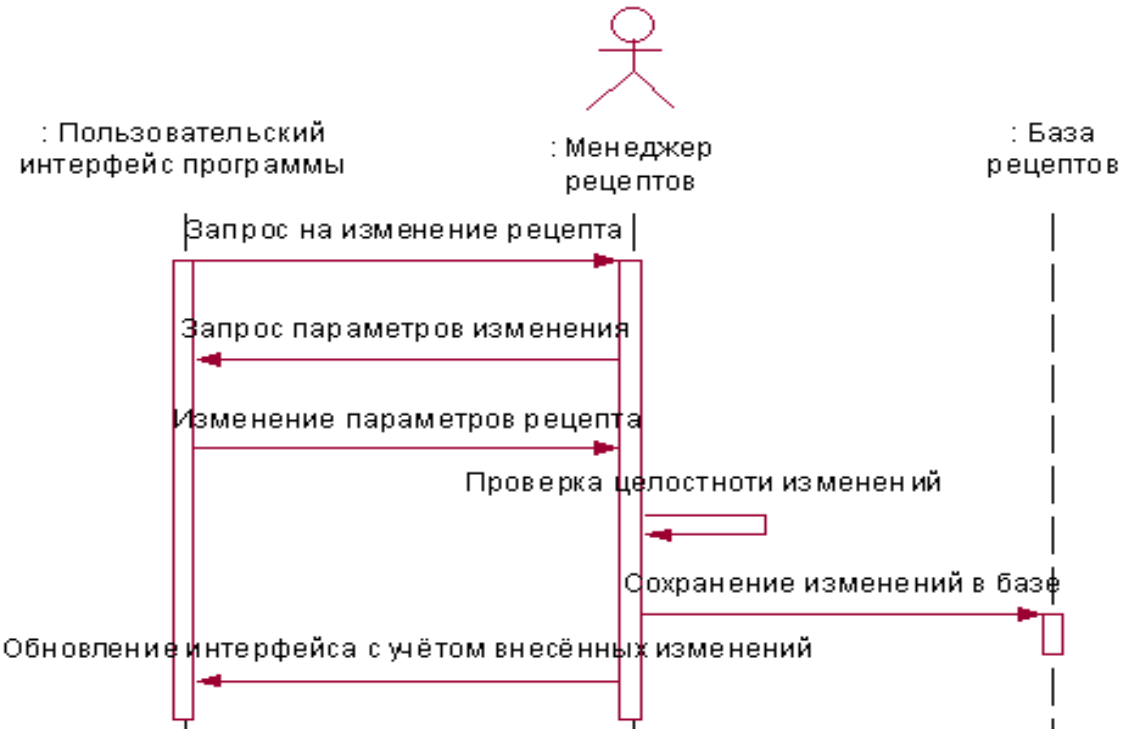

Рис. 6.39. Модификация рецепта

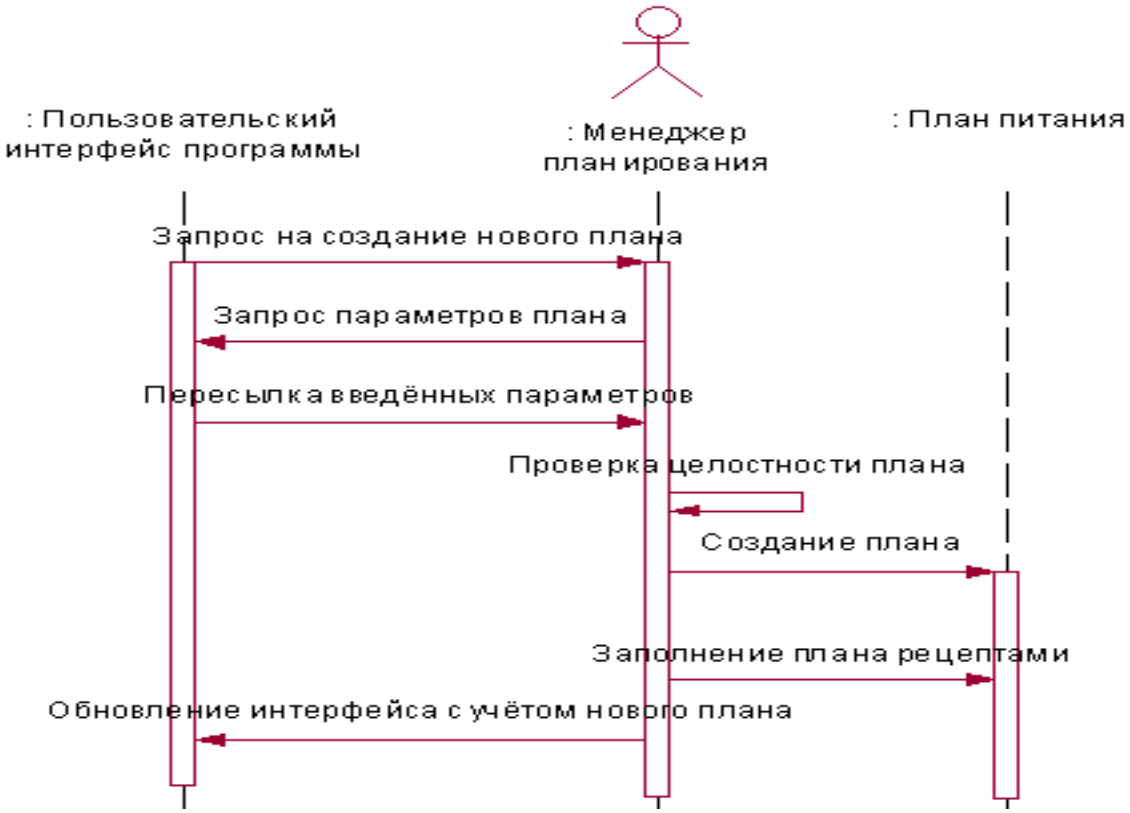

Рис. 6.40. Создание нового плана питания

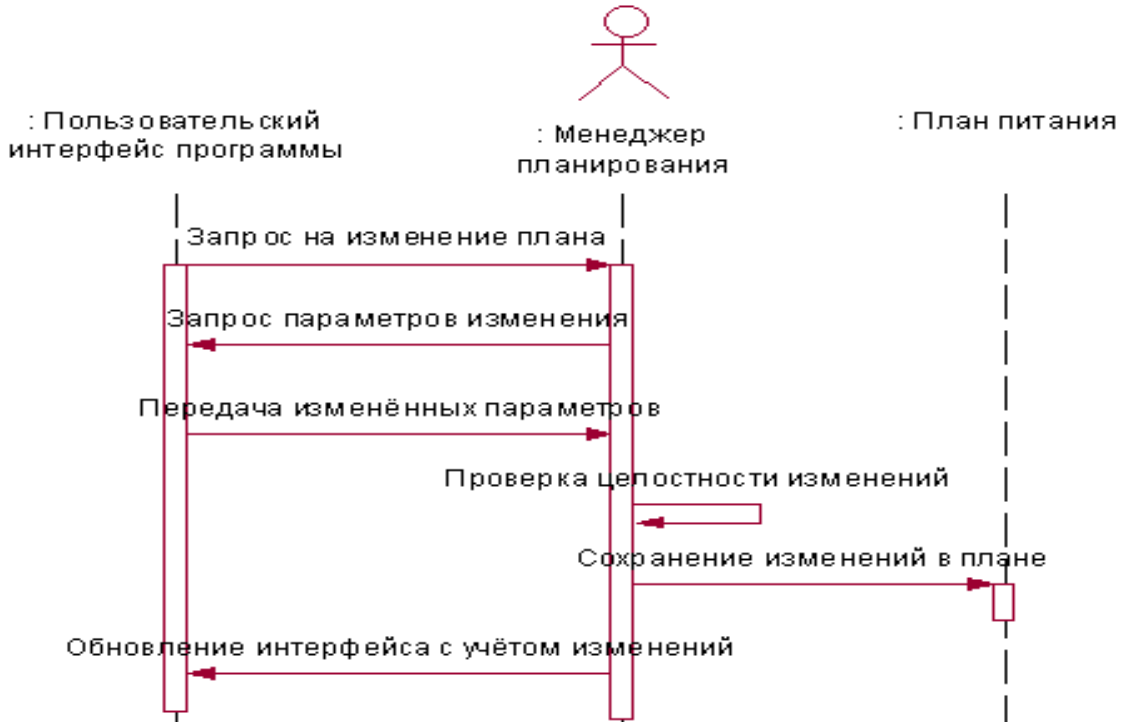

Рис. 6.41. Модификация текущего плана

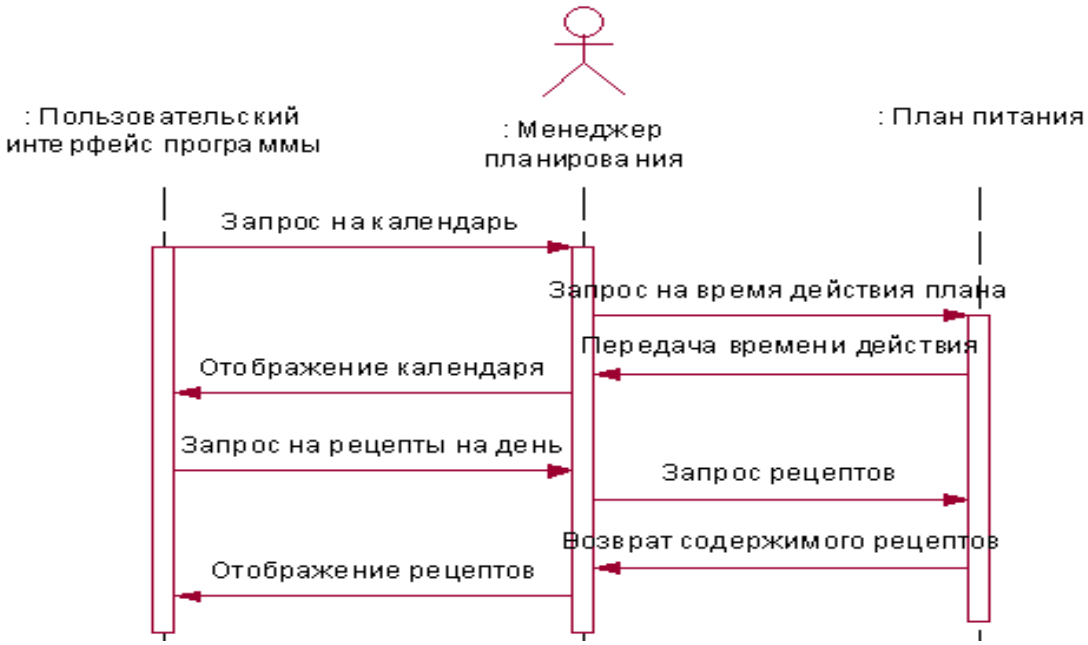

Рис. 6.42. Просмотр текущего плана

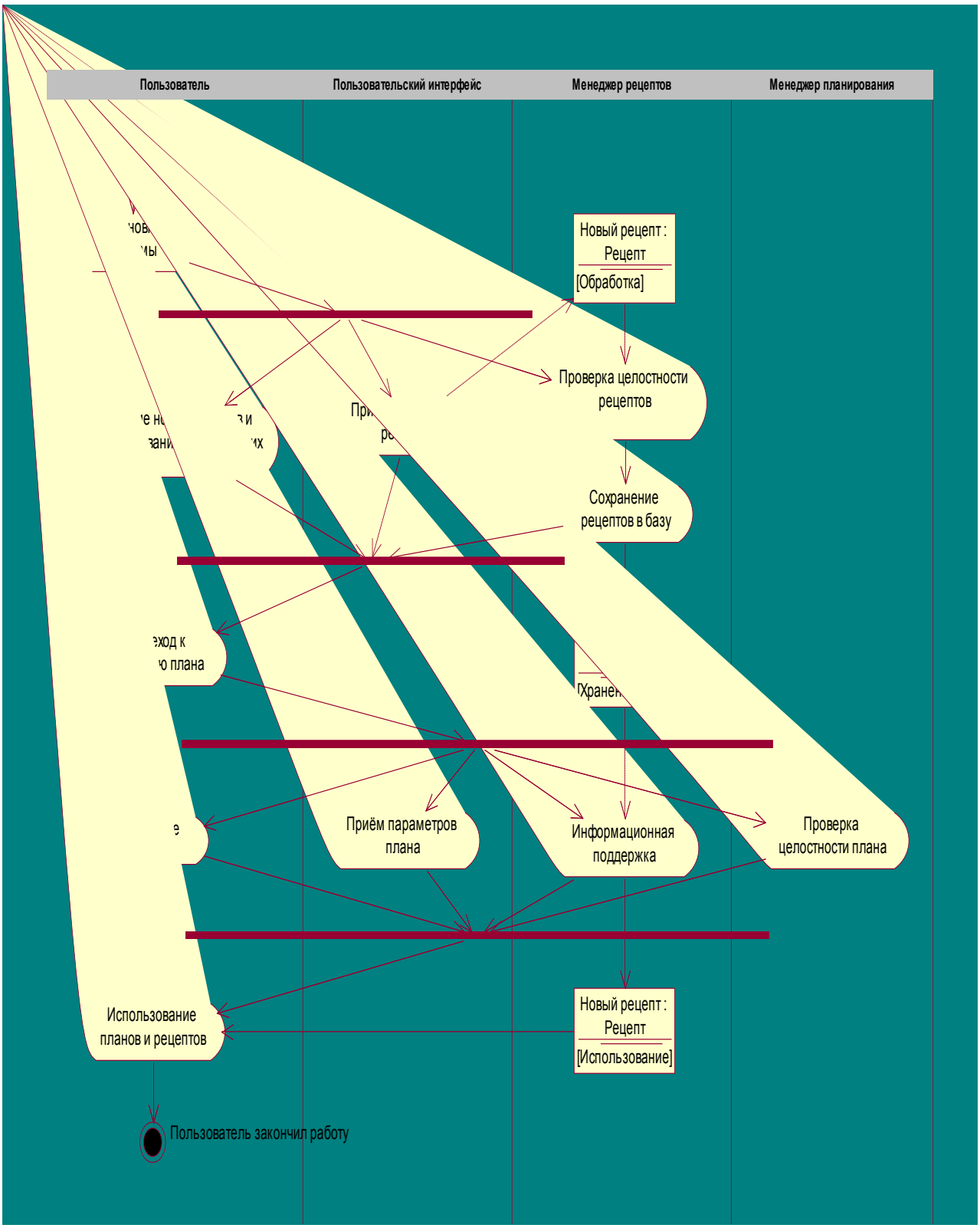

Рис. 6.43. Диаграмма деятельности

Все диаграммы, связанные с планированием питания, отвечают требованиям предметной области и дополнены особенностями планируемой реализации.

На основании требований, предъявляемых к проектируемой среде, возможно создание диаграмм *конечных автоматов* (State Machine Diagram): диаграмм *деятельности* и диаграмм *состояний.*

На рис. 6.43 приведена *диаграмма деятельности* **Пользователя**. Здесь показан алгоритм взаимодействия основных компонент системы в процессе функционирования КП.

Первым этапом деятельности *Пользователя* является установка системы. При этом база рецептов пока пуста.

Далее *Пользователь* заполняет базу необходимыми для создания плана рецептами. Пользовательский интерфейс осуществляет прием параметров нового рецепта, а также позволяет вносить изменения в уже созданные рецепты. Менеджер рецептов осуществляет проверку данных, поступивших от *Пользователя*, на целостность. После этого идет сохранение рецепта в базу данных.

После того как заполнение базы рецептов окажется достаточным для разработки полноценных планов питания, *Пользователь* может переходить к их созданию. На данном этапе приложение предоставляет *Пользователю* возможность, комбинируя имеющиеся рецепты, создавать план, а также задавать необходимые параметры этого плана. На основе информации предоставляемой *Менеджером рецептов*, *Менеджер планирования* осуществляет проверку плана на целостность.

Естественным продолжением деятельности пользователя является использование созданных планов и рецептов.

На *диаграмме состояний* (рис. 6.44) представлен жизненный цикл *объекта* “**Рецепт**”. Он включает в себя *три состояния* – **Обработка, Хранение и Использование**.

Состояние *обработки* – это создание и редактирование рецепта. Такие состояния объекта, как *хранение* и *обработка*, могут следовать одно за другим, пока пользователь вносит изменения в рецепт.

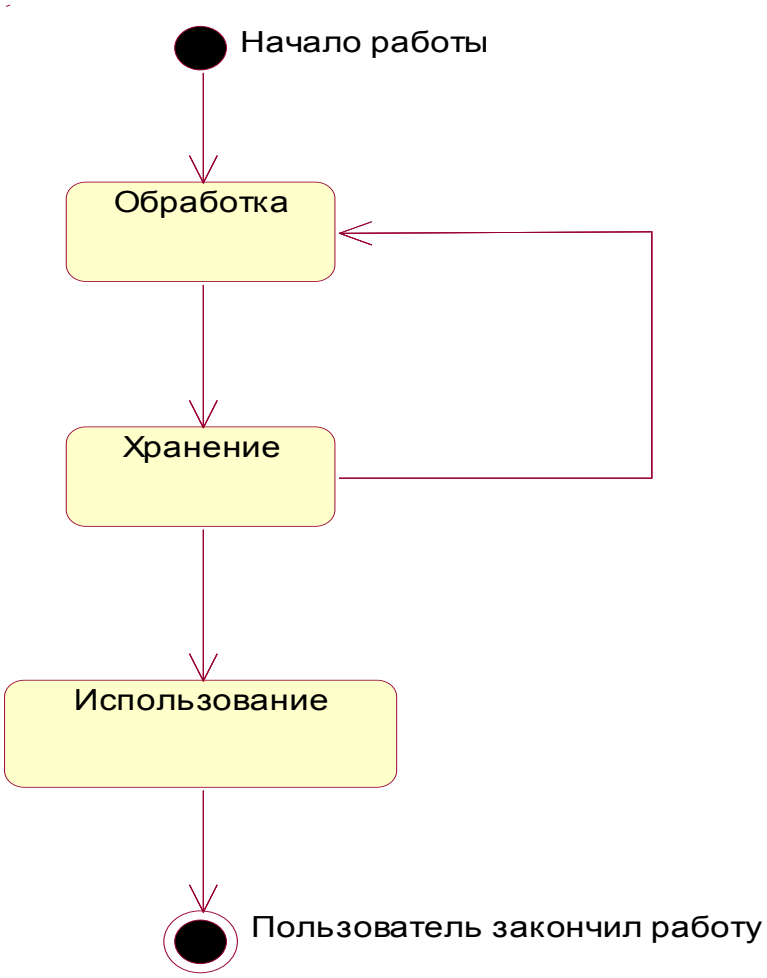

Рис. 6.44. Диаграмма состояний

Проектирование

Для проектирования архитектуры системы мы будем использовать *диаграммы классов* (Class Diagram). На рис. 6.45 показана возможная реализация такой диаграммы для вариантов использования, описанных выше.

Как видно из диаграммы классов, база рецептов реализована в виде иерархического набора экземпляров класса **Chapter,** идентифицируемых по имени. Этот класс представляет собой раздел “кулинарной книги”, которой мы управляем. В зависимости от желания Пользователя можно организовывать любое разбиение

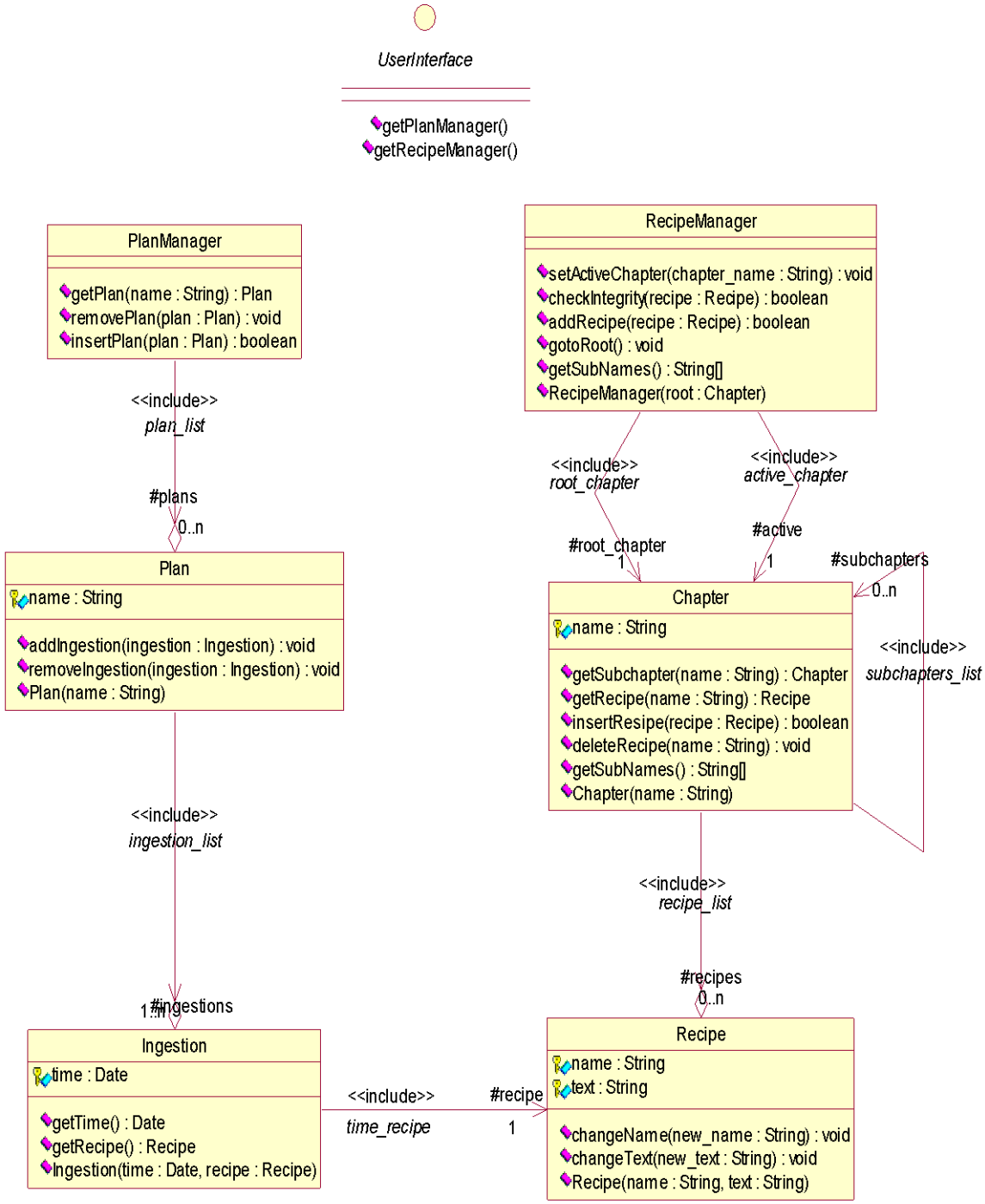

Рис. 6.45. Диаграмма классов

рецептов. Например, в “Книге” могут быть разделы “Грузинская кухня”, “Европейская кухня”, “Балканские блюда” и т. д. Каждый (или некоторые) из разделов может разбиваться на подразделы: “Закуски”, “Первые блюда” и т. д.

Каждый экземпляр хранит список (возможно, пустой) ссылок на все рецепты, которые он включает, а также список (возможно, пустой) ссылок на все свои подразделы. Предусмотрена возможность добавления и удаления

рецепта, получения подраздела и рецепта по имени. Операция добавления рецепта может быть неуспешной (рецепт с таким именем уже существует). Поэтому метод **insertRecipe** возвращает логическое значение, которое говорит, насколько успешно прошло добавление (*true* – успешно, *false* – неуспешно).

Сам рецепт (Recipe) есть совокупность имени рецепта и его содержимого. Возможно изменение, как имени, так и текста рецепта.

Менеджер рецептов **RecipeManager** содержит указатели на раздел *верхнего* уровня (**root_chapter** – вся "Кулинарная книга") и на *текущий* раздел просмотра и "умеет" обеспечивать просмотр базы рецептов при помощи методов **setActiveChapter**, **gotoRoot** и **getSubNames** и её модификацию с помощью методов **addRecipe** и **checkIntegrity**.

План питания представляет собой список рецептов с указанием времени использования каждого рецепта. Для удобства эта структура (рецепт плюс время его использования) выделена в отдельный класс – **Ingestion**. При этом сам **план** (класс Plan) содержит массив таких структур и обеспечивает простейший к нему доступ с помощью методов **addIngestion и removeIngestion**.

Менеджер планирования представляет собой список доступных планов питания. Пользователь имеет возможность добавить новый план, получить план по имени и удалить существующий план.

Кроме того, интерфейс **UserInterface** предоставляет доступ как к менеджеру планирования, так и к менеджеру рецептов.

Генерация кода

Одним из мощных свойств Rational Rose является возможность генерации программного кода после построения модели.

Общая последовательность действий, которые необходимо выполнить для генерации кода, состоит из следующих **шести** этапов:

1. **Проверка модели** (не зависит от языка генерации кода).
2. **Создание компонент для реализации классов.**
3. **Отображение классов на компоненты.**
4. **Установка свойств генерации кода.**
5. **Выбор классов, компонент или пакетов.**
6. **Генерация кода.**

Особенности выполнения каждого из этапов могут изменяться в зависимости от выбранного языка.

В среде Rational Rose *предусмотрено задание* достаточно большого числа *свойств* для характеристики классов, пакетов, компонент и проекта в целом. О возможностях настройки свойств под свои потребности подробно изложено ниже в данной главе (пункт 6.5). В примере используются *стандартные настройки* Rational Rose.

Рассмотрим более подробно последовательность действий для каждого этапа.

Проверка модели

Для проверки всей модели выберите в меню **Tools->Java->Syntax Check**. Согласно этой опции будут проверены все *классы*, *переменные*, *методы* и *операторы* на соответствие синтаксису *Java*.

В случае обнаружения ошибок будет сгенерирован **Log** – файл ошибок. К нему можно перейти с помощью **Window->Log**.

Создание компонент

Для того чтобы запустить процесс генерации кода для класса, необходимо создать его реализацию в виде компонент. Для этого в разделе **Component View** необходимо нажать *правую* кнопку мыши и в появившемся *выпадающем* меню выбрать **New->Component**. Затем, установив курсор мыши на вновь созданный компонент, нажать правую кнопку и выбрать пункт **Open Specification**. В диалоге Open Specification *изменить* поле **Language** на *Java*.