Анализ и управление требованиями

Часть 3
Управление. Моделирование. Прототипирование

Управление требованиями

представляет собой:

• систематический подход к выявлению, организации и документированию требований к системе;

• процесс, устанавливающий соглашение между заказчиками и разработчиками относительно изменения требований к системе и обеспечивающий его выполнение.

Управление требованиями

преследует цели:

достичь соглашения с заказчиком и пользователями о том, что система должна делать;

улучшить понимание требований к системе со стороны разработ чиков;

очертить границы системы;

определить базис для планирования

определить приоритеты

ПО для управления требованиями

IBM Rational Requisite Pro

Программное обеспечение Rational представляет лучшие практические методы определения требований и управления ими, которые обеспечивают экономию времени и средств, помогая в решении следующих задач:

- Сокращение объема доработок и ускорение выхода на рынок благодаря совместной работе с заинтересованными лицами.
- Повышение производительности труда за счет контроля над изменениями в требованиях и управлении ими.
- Минимизация расходов и рисков за счет оценки влияния происходящих изменений.
- Демонстрация соответствия требований благодаря полному отслеживанию требований.

ПО для управления требованиями

IBM Rational/Telelogic DOORS

IBM Rational/Telelogic DOORS — семейство решений для управления требованиями и создания сложных наукоемких изделий (авиа, судостроение, поезда, ракеты, автомобили т.п.).

Из Telelogic DOORS можно получить следующую информацию:

- Статус выполнения работ по каждому требованию отдельно, а также по группе требований.
- Статус работы над всем проектом.
- Ответственное лицо для каждого требования или группы требований.
- Историю изменений требования.
- Ресурсы, которые потребуются для реализации требования еще до его внедрения в проект.
- Связь между требованиями заказчика, пунктами технического задания, программами верификации, тестирования и задачами управления проектом.
- Класс, модель или чертеж, в котором конкретное требование реализовано.

ПО для управления требованиями

Borland Caliber RM

Borland Caliber RM – это корпоративная система управления требованиями, которая облегчает совместную работу, что позволяет группам разработчикам подходить к вехам проекта вовремя и с запланированными затратами.

Обладает следующими функциональными возможностями:

Централизованное хранилище требований для всех проектов, разрабатываемых IT-компанией.

Адаптируемость — Caliber RM можно сконфигурировать для использования в любом проекте, что повышает эффективность процесса управления требованиями.

Трассируемость требований — открытая архитектура Caliber RM позволяет связать требования с другими артефактами на всех стадиях жизненного цикла программного продукта.

Поддержка большого числа клиентов — Caliber RM прекрасно интегрируется с такими системами разработки, как Microsoft Visual Studio, Eclipse на платформе Windows.

Интеграция с другими продуктами Borland для поддержки полного жизненного цикла программного продукта.

ПО для управления требованиями

Sybase PowerDesigner

OpenSource Requirements Management Tool

RequirementsWin и другие

Средства управления требованиями обладают различными возможностями в зависимости от подхода разработчика.

Стандартными можно считать две из них:

• выделение требований непосредственно в документах различного формата с сохранением ссылки на исходный текст;

• отслеживание зависимостей между требованиями.

Системные требования

определение

Системные требования состоят из полного списка конкретных свойств и функциональности, которую должна иметь программа, сформулированных в подробностях

Назначение: представление системы

Внешнее представление, когда моделируется окружение или рабочая среда системы

Описание поведения системы, когда моделируется ее поведение.

Описание структуры системы, когда моделируется системная архитектура или структуры данных, обрабатываемых системой.

Модель является абстракцией системы и легче поддается анализу, чем любое другое представление этой системы

Типы системных моделей

Модель обработки данных. Диаграммы потоков данных показывают последователь ность обработки данных в системе.

Композиционная модель. Диаграммы "сущность-связь" показывают, как системные сущности составляются из других сущностей

Архитектурная модель. Эти модели показывают основные подсистемы, из которых строится система.

Классификационная модель. Диаграммы наследования классов показывают, какие объекты имеют общие характеристики

Модель "стимул-ответ". Диаграммы изменения состояний показывают, как система реагирует на внутренние и внешние события.

Системные модели (обобщение)

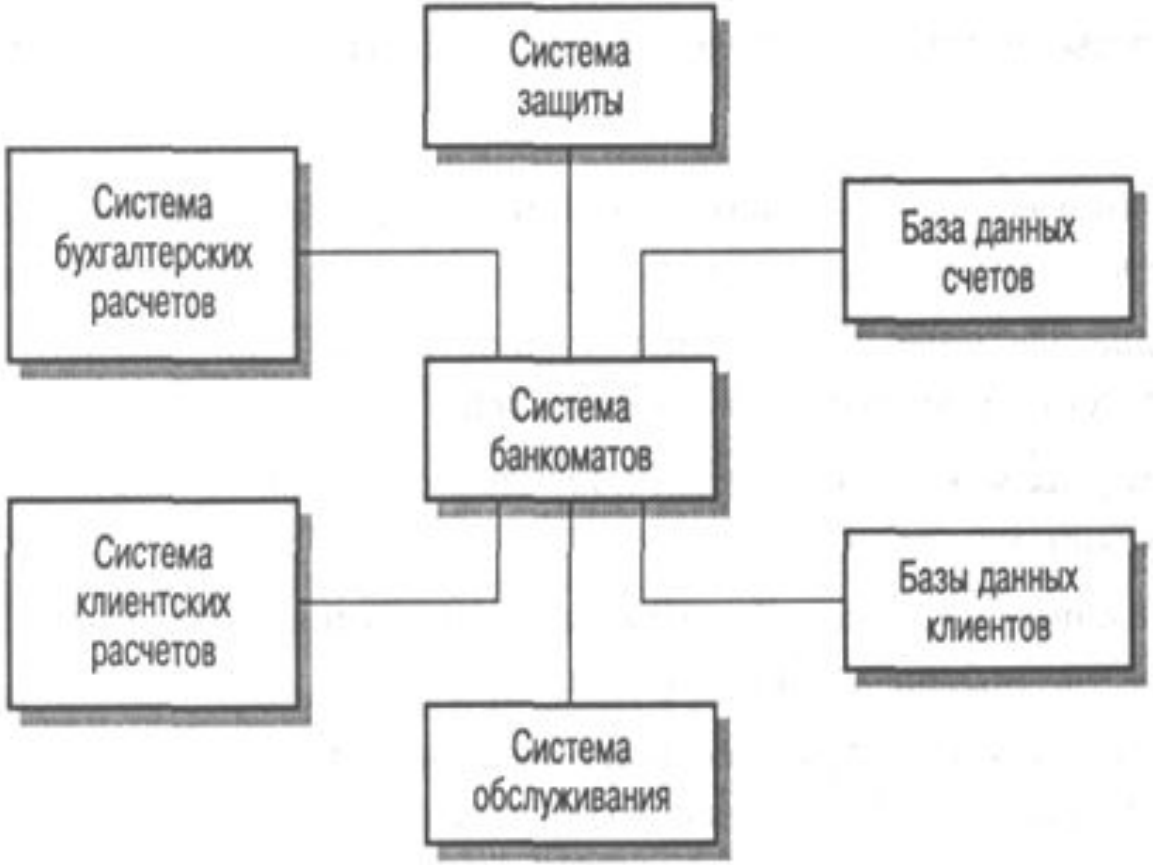
модели системного окружения,

поведенческие модели

модели данных

Модели системного окружения

служат для определения границ системы и строятся на ранних этапах выявления требований. Трудоемкость построения модели зависит от характера разрабатываемого приложения.



Пример модели окружения

Простые структурные модели обычно дополняются моделями других типов, например **моделями процессов**, которые показывают взаимодействия в системе, или **моделями потоков данных**, которые показывают последовательность обработки и перемещения данных внутри системы и между другими системами в окружающей среде

Поведенческие модели

используются для описания общего поведения системы

1. Модель поведения должна быть *компактной и обозримой*, чтобы служить средством общения между людьми в процессе разработки системы и для обмена идеями.
2. Средства моделирования поведения *должны быть знакомы и привычны* для большинства заинтересованных лиц.
3. Модель поведения *не должна зависеть от особенностей реализации* конкретных систем, инструментальных средств, технологий, чтобы не сужать область ее применения

Поведенческие модели

1. Модели потоков данных
2. Модель конечного автомата
3. Модели данных

Модели потоков данных (МПД)

либо показывают функциональную структуру системы, где каждое преобразование данных соответствует одной системной функции.

либо используют для описания потоков данных в рабочем окружении системы. Такая модель показывает, как различные системы и подсистемы обмениваются информацией. Подсистемы окружения не обязаны быть простыми функциями.

Представление МПД

Для описания модели потоков данных используются **диаграммы потоков данных** (data flow diagram, DFD), во всем многообразии нотаций которых, можно выделить основные компоненты:

- *процесс* (трансформационный процесс, системная функция, обрабатывающая единица) – механизм, описывающий способ получения выходных данных из входных данных;
- *поток данных* – определяющий перемещение данных (материалов) между процессами;
- *хранилище* – места хранения данных (базы данных, например);
- *внешние сущности* – объекты внешнего (по отношению к системе) мира.

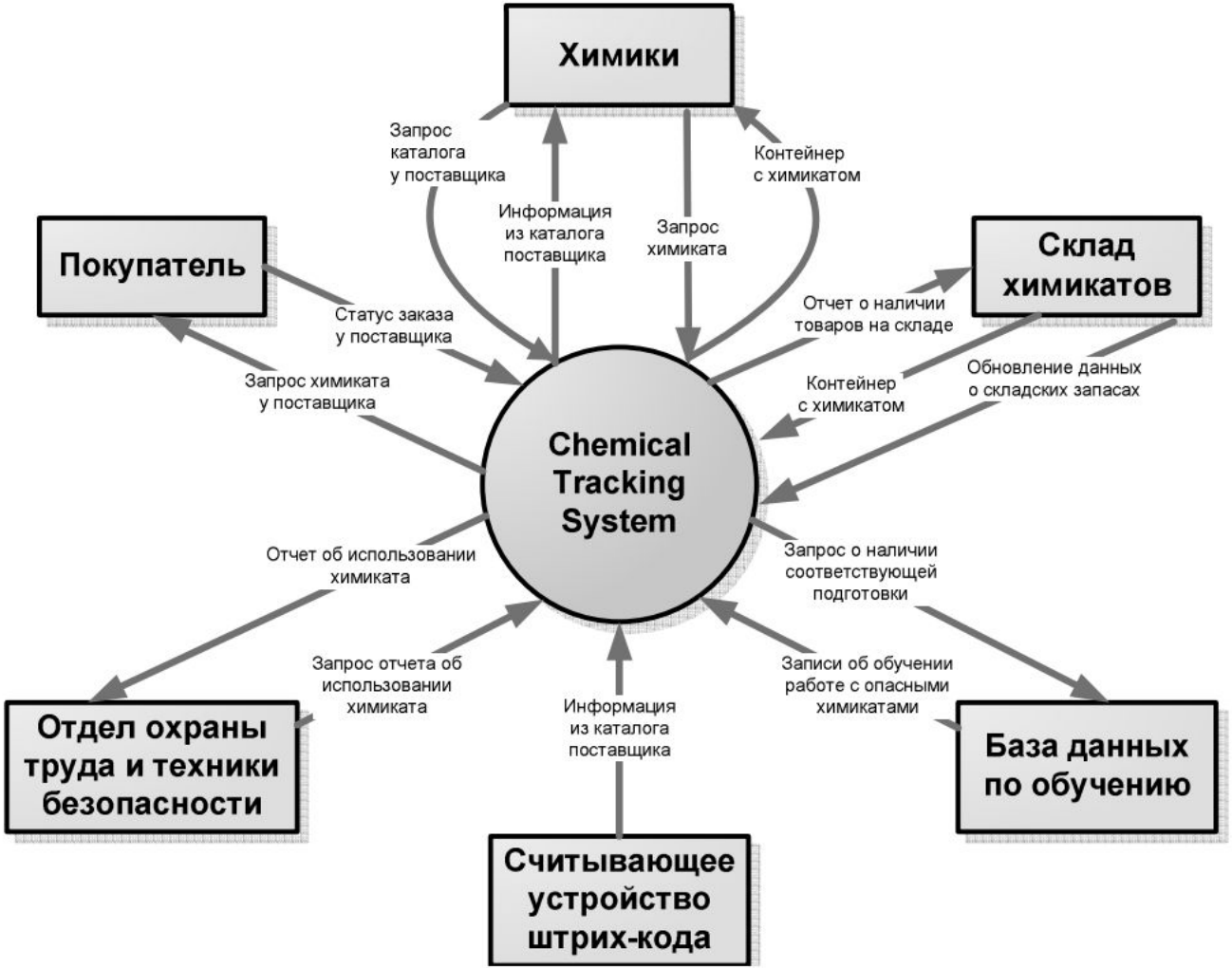
Диаграмма потоков данных комплекса CASE-средств

Представление МПД

Контекстная диаграмма содержит внешние, по отношению к системе, сущности, которые являются источниками или потребителями системных потоков данных

Выделив важнейшие процессы системы, можно разработать диаграмму потока данных уровня 0, в которой, кроме внешних сущностей и системных потоков данных, определенных в контекстной диаграмме, содержит эти процессы и те потоки данных, которыми они обмениваются. Каждый процесс из диаграммы нулевого уровня может быть детализирован в виде диаграммы следующего уровня (уровень 1), содержащего всю его функциональность.

Контекстная диаграмма



Ценность МПД

позволяют:

- представить бизнес-процесс или операцию, выполняемую системой, в виде совокупности этапов;
- проследить и документировать перемещение данных по системе, помогая понять этот процесс;
- учитывая их простоту и понятность, использовать модели для общения с пользователями, занятыми в разработке требований;
- описывать (при проектировании) принятые решения.

Правила разработки МПД

При разработке диаграмм потоков данных следует придерживаться следующих общих правил:

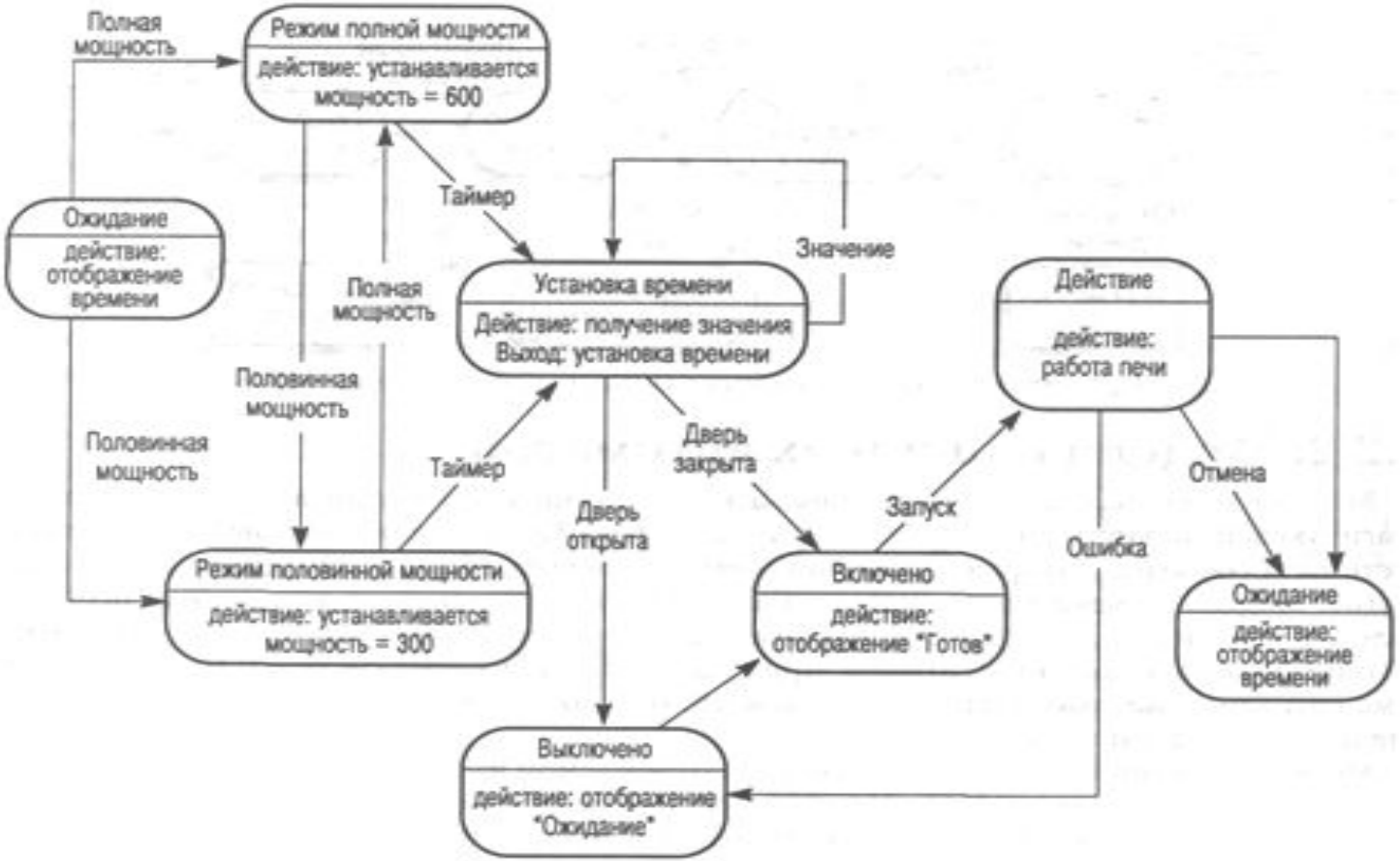
- размещать на диаграмме не более 7 – 10 процессов;
- не загромождать диаграммы несущественными на данном уровне деталями;
- декомпозицию потоков данных выполнять параллельно с декомпозицией процессов;
- однократно определять функционально идентичные процессы и ссылаться на них, на нижних уровнях.

Модели конечных автоматов

используются для моделирования поведения системы, реагирующей на внутренние или внешние события.

Такая модель показывает состояние системы и события, которые служат причиной перехода системы из одного состояния в другое.

ДИАГРАММА СОСТОЯНИЙ (UML)



Пример модели конечных автоматов

Модели данных

Наиболее широко используется моделирование данных типа “сущность – связь – атрибут”, определяющее структуру данных, их атрибуты и отношения между ними.

Для разработки моделей данных, обеспечивающих стандартный способ определения данных и отношений между ними, служат диаграммы “сущность-связь” (Entity-Relationship Diagrams, **ER-диаграммы).**

ER - диаграммы

Под *сущностью* (entity) в ER-диаграммах понимается множество экземпляров реальных или абстрактных объектов (людей, предметов, событий и т.д.). Любой объект системы может быть представлен только одной уникально идентифицированной сущностью, имя которой должно отражать тип (класс) объектов, а не его конкретный экземпляр.

Связь (relation) – это именованное отношение между двумя и более сущностями, а *атрибут* (attribute) – любая характеристика сущности, причем множество значений всех атрибутов должно однозначно идентифицировать экземпляр сущности.

СЛОВАРЬ!

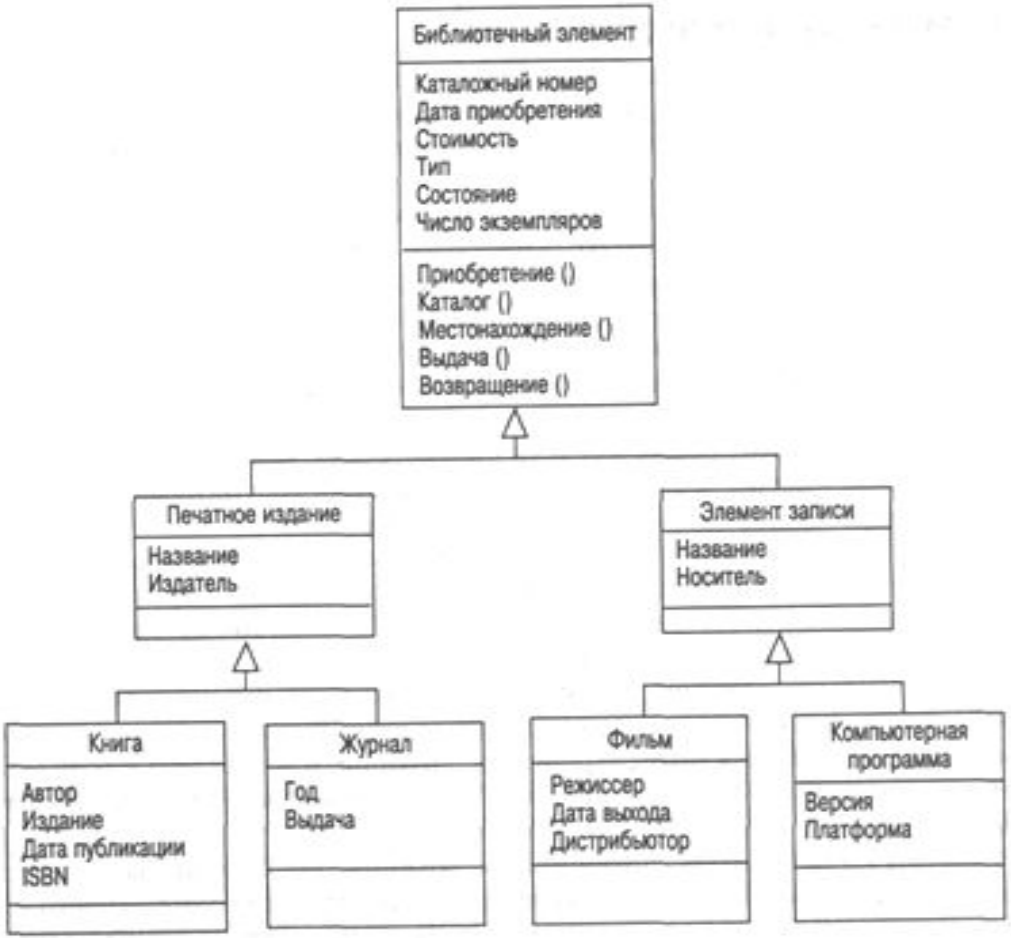
Объектные модели

Объектные модели, могут использоваться как для представления данных, так и для процессов их обработки. В этом отношении они объединяют модели потоков данных и семантические модели данных. Они также полезны для классификации системных сущностей и могут представлять сущности, состоящие из других сущностей.

Класс объектов - это абстракция множества объектов, которые определяются общими атрибутами и сервисами (операциями).

Объекты - это исполняемые сущности с атрибутами и сервисами класса объектов. Объекты представляют собой реализацию класса. На основе одного класса можно создать много различных объектов.

Диаграмма классов

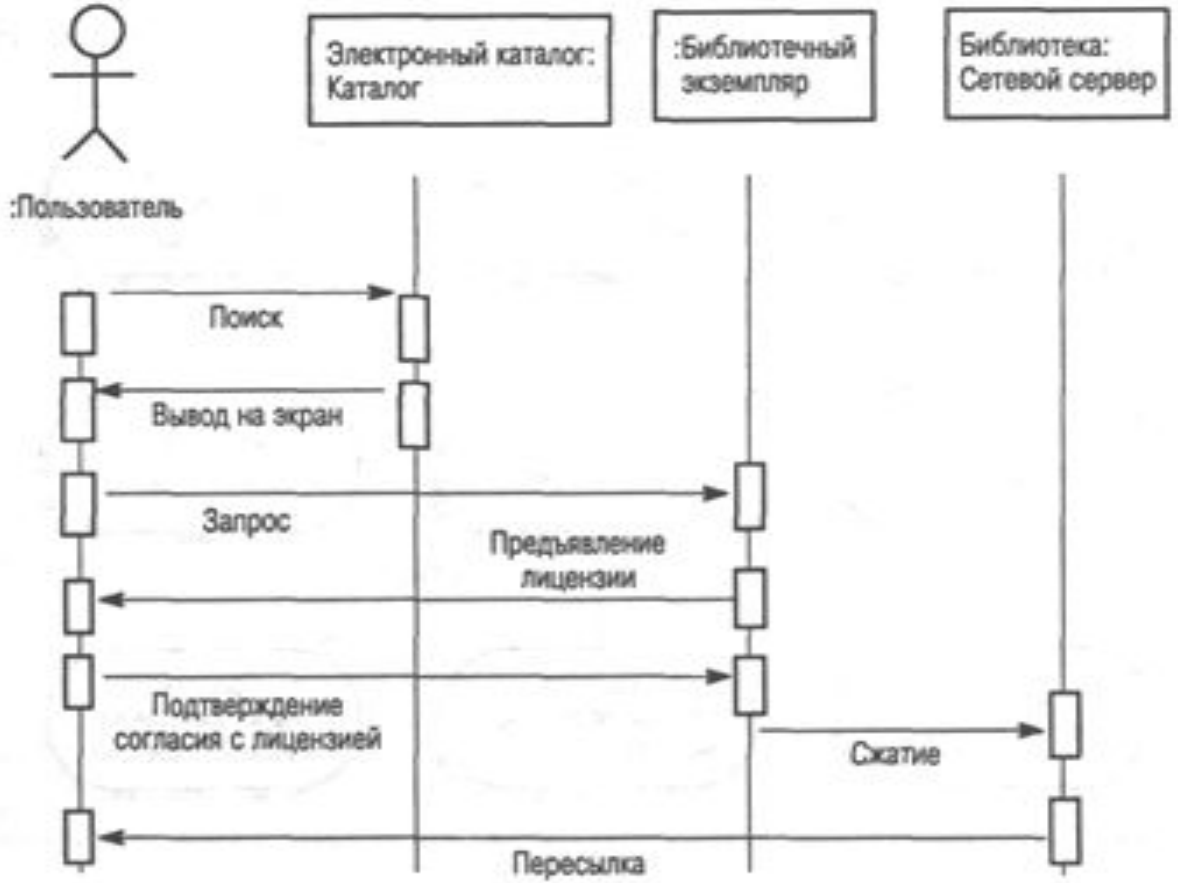


Модели поведения объектов

В UML поведение объектов моделируется посредством сценариев, которые основаны на вариантах использования.

Диаграммы последовательности.

В верхней части расположены объекты. Операции обозначаются помеченными стрелками, а последовательность операций читается сверху вниз.



Пример диаграммы последовательности

Прототип

это начальная версия программной системы, которая используется для демонстрации концепций, заложенных в системе, проверки вариантов требований, а также поиска проблем, которые могут возникнуть как в ходе разработки, так и при эксплуатации системы, чтобы пользователи могли начать экспериментировать с ним как можно раньше

Разработка прототипа позволяет сделать требования более реальными, помогает заинтересованным лицам прийти к общему пониманию требований, ускоряет процесс разработки и анализа.

Прототип полезен для решения задач:

1. *Разработка требований.*
2. *Проверка требований.*
3. *Разработка взаимодействия с пользователем.*
4. *Спецификация системных требований.*

Виды прототипов

1. Горизонтальные (поведенческие)
2. Вертикальные (структурные)

Разработка прототипов

1. Экспериментальное прототипирование (пассивный горизонтальный прототип)
2. Эволюционное прототипирование
    a. 1.*Ускорить разработку программной системы*. В некоторых случаях быстрая разработка и поставка системы, удобство и простота использования перевешивают факт ее функциональной неполноты.
    b. 2.*Участвовать пользователям в процессе разработки*. Взаимодействие пользователя с системой – это гарантии более полного учета их требований.

Риски прототипирования

1. *Заинтересованные в проекте лица могут принять работающий прототип за начальную версию системы*.
2. *Пользователи системы могут использовать работающий прототип для анализа и критики интерфейса*

3. *Пользователи системы могут использовать работающий прототип для определения характеристик качества системы*

4. *Разработчики могут много сил и средств потратить на разработку прототипа*.

Например, для уточнения требований и пользовательского интерфейса могут быть использованы горизонтальные прототипы, для разработки архитектуры – вертикальные прототипы, а для проектирования – эволюционные прототипы