Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Специальность: «Программное обеспечение информационных технологий»

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Технология разработки программного обеспечения»

На тему: «Информационная система кинотеатра»

Студент-заочник 2 курса

Группы № 581072

ФИО: Мордасова Ксения Александровна

Адрес г. Минск,

пр. Газеты «Правда»

д. 14, к. 3, кв.64

Тел. 8 029 619 24 17

Проверила: Бакунова О.М.

Минск, 2017

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 ПОСТАНОВКА ЗАДАЧИ 4

1.1 Описание предметной области 4

1.2 Цели и задачи проектирования 4

2 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ. 6

3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 8

3.1 Диаграмма вариантов использования 9

3.2 Диаграмма классов 11

3.3 Диаграмма деятельности 12

3.4 Диаграмма коопераций 14

3.5 Диаграмма последовательности 16

3.6 Диаграмма развёртывания 17

3.7 Диаграмма компонентов 19

4 ГЕНЕРАЦИЯ КОДА 20

ЗАКЛЮЧЕНИЕ 30

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31

**ВВЕДЕНИЕ**

Современная жизнь немыслима без эффективного управления. Важной категорией являются системы обработки информации, от которых во многом зависит эффективность работы любого предприятия или учреждения.

Для принятия обоснованных и эффективных решений в производственной деятельности, в управлении экономикой и в политике современный специалист должен уметь с помощью компьютеров и средств связи получать, накапливать, хранить и обрабатывать данные, представляя результат в виде наглядных документов.

Спецификация требований программного обеспечения (англ. Software Requirements Specification, SRS) — законченное описание поведения программы, которую требуется разработать.

Включает ряд пользовательских сценариев (англ. use cases), которые описывают все варианты взаимодействия между пользователями и программным обеспечением.

Пользовательские сценарии являются средством представления функциональных требований. В дополнение к пользовательским сценариям, спецификация также содержит нефункциональные требования, которые налагают ограничения на дизайн или реализацию (такие как требования производительности, стандарты качества, или проектные ограничения).

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению, в результате анализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования программного обеспечения (ПО). Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Прежде, чем начинать разработку приложения необходимо определить требуемую функциональность создаваемого программного обеспечения. Для этого воспользуемся uml-моделированием.

С помощью моделирования различных областей деятельности можно достаточно эффективно анализировать “узкие места” в управлении и оптимизировать общую схему бизнеса.

IDEF0 - методология функционального моделирования. В ходе реализации программы интегрированной компьютеризации производства (ICAM), предложенной в свое время ВВС для аэрокосмической промышленности США, была выявлена потребность в разработке методов анализа взаимодействия процессов в производственных системах. Для удовлетворения этой потребности была разработана методология IDEF0 (Integrated Definition Function Modeling), которая в настоящее время принята в качестве федерального стандарта США. Методология успешно применялась в самых различных отраслях, продемонстрировав себя как эффективное средство анализа, проектирования и представления деловых процессов. В настоящее время методология IDEF0 широко применяется не только в США, но и во всем мире.

Темой контрольной работы является разработка программного продукта «Информационная система кинотеатра». Кинотеатр предоставляет услуги культурного отдыха, это один из самых распространенных видов отдыха в любой стране. Ежегодно в прокат выходят тысячи фильмов, записи о которых необходимо где-нибудь хранить. Для успешного поддержания работы кинотеатра необходимо современная информационная система для обработки поступающих заказов в кинотеатр.

Для реализации поставленной цели необходимо решение следующих задач:

* разработка логической модели программного средства;
* создание простого и понятного интерфейса;
* создание единой базы данных для работы программы;
* обеспечение целостности данных при работе с базой данных;
* возможность редактирования, изменение, добавление данных в программу;
* предоставление возможности работы с базой данной простому пользователю, не являющимся его администратором;

# **1 ПОСТАНОВКА ЗАДАЧИ**

## **Описание предметной области**

Основной оценкой качества обслуживания в кинотеатре является скорость продажи билетов. Если в кассе выдают билеты качественно и быстро, то в выигрыше остаются все - и клиент, которому понравится уровень сервиса и администрация кинотеатра, так как довольные покупатели чаще всего превращаются в постоянных.

В настоящее время потребность в автоматизированных информационных системах идет на подъем. Для того чтобы обеспечить желание отдыхающих насладиться отдыхом создано и продолжает создаваться огромное количество кинотеатров. Залогом успешной деятельности кинотеатра является максимально эффективная организация его работы, предоставление клиентам наиболее широкого спектра услуг, качество и надежность их предоставления.

Программное средство (ПС), должно обеспечить возможность ведения учета информации о показах, которые будут проходить в данном кинотеатре, а также ведения базы данных фильмов, показ которых будет идти в кинотеатре и соответственно оформлением билетов.

* 1. **Цели и задачи проектирования**

Требуется создать программное средство, которое бы обеспечивала автоматизированную информационную систему кинотеатра. Суть данной программы в том, чтобы расширить возможности кинотеатра, а также автоматизировать его основные функции.

Необходимо предоставить возможность добавления, редактирования, удаления и поиска данных по базе. Данные, которые зависят друг от друга, должны быть корректно сопоставлены.

Разрабатываемая программа должна обеспечивать следующее:

* точность информации о фильмах, показах, сеансах;
* возможность доступа к требуемой информации;
* ведение базы данных.

Требования, предъявляемые к данной программе:

* организация удобного пользовательского интерфейса;
* возможность постоянного обновления основных таблиц базы данных;
* возможность удаления записей из основных таблиц;
* возможность быстрой сортировки и поиска данных.

К входным данным относятся следующие пункты:

* фильмы, которые будут показываться в кинотеатре;
* показы.

Выходными данными являются:

* информация о фильмах, которые идут в кинотеатре;
* билет в кино (код показа, название фильма, номер места, номер ряда, дата показа, и дата продажи).

К постоянным данным относятся:

* сведения о сеансах.

Разработанное программное обеспечение позволит быстро и легко как вести базу данных, так и формировать отчеты для дальнейшего развития кинотеатра. Выбор СУБД и языка программирования позволили создать компактную реляционную модель базы данных отвечающей последним инновационным требованиям. Внешний дизайн каждой формы должен быть ориентирован на интуитивный интерфейс и позволяет легко и быстро получать интересующую в данный момент информацию.

1. **ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ**

Субъектом моделирования является программное средство «Автоматизированная информационная система кинотеатра», цель – описать функциональность работы программного средства, точка зрения– посетитель, администратор.

IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ).

В основе IDEF0 методологии лежит понятие блока, который отображает некоторую бизнес-функцию. Четыре стороны блока имеют разную роль: левая сторона имеет значение "входа", правая - "выхода", верхняя - "управления", нижняя - "механизма" (рисунок 2.1)

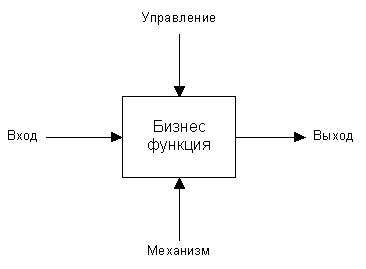


Рисунок 2.1 – Стрелки управления функционального блока

В IDEF0 реализованы три базовых принципа моделирования процессов:

* принцип функциональной декомпозиции;
* принцип ограничения сложности;
* принцип контекста.

Принцип контекстной диаграммы. Моделирование делового процесса начинается с построения контекстной диаграммы. На этой диаграмме отображается только один блок - главная бизнес-функция моделируемой системы. Главная бизнес-функция разрабатываемой системы – КИНОТЕАТР, так как программное средство реализует почти все функции кинотеатра: выбор фильма, просмотр доступных сеансов, выбор времени и мест в зале, покупка билета. Исключение составляет сам показ фильма, для этого действия нет необходимости в использовании информационной системы.

Входные данные:

* информация о клиентах (предпочтения, платежная информация, возраст, пол и др.);
* информация о сеансах (название фильма, дата показа, время показа, возвратной рейтинг, цена и др.);
* авторизация.

Управление:

* законодательство РБ;
* министерство торговли РБ
* общество защиты прав потребителей.

Механизмы:

* администратор;
* посетитель.

Выходные данные:

* измененная БД;
* отчетная документация;
* информация об ошибках.

Контекстная диаграмма модели представлена на рисунке 2.2

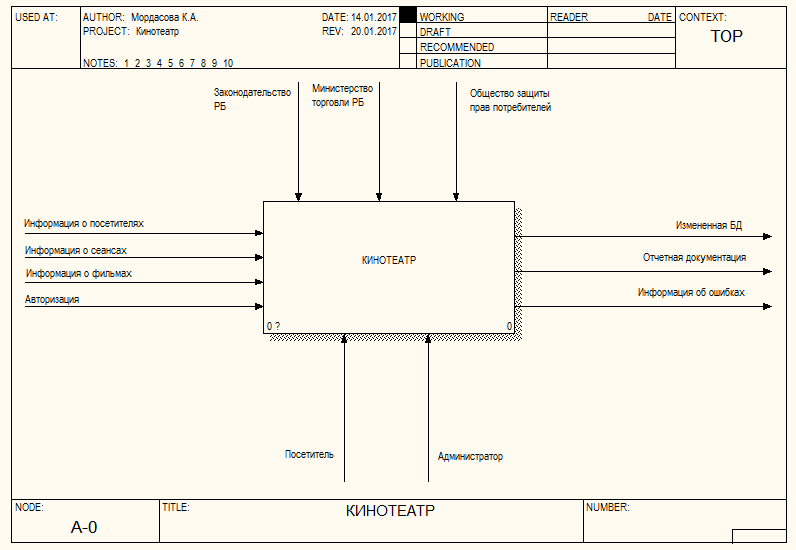


Рисунок 2.2 – Контекстная диаграмма

Принцип ограничения сложности. При работе с IDEF0 диаграммами существенным является условие их разборчивости и удобочитаемости. Суть принципа ограничения сложности состоит в том, что количество блоков на диаграмме должно быть не менее двух и не более шести. Практика показывает, что соблюдение этого принципа приводит к тому, что функциональные процессы, представленные в виде IDEF0 модели, хорошо структурированы, понятны и легко поддаются анализу.

Принцип функциональной декомпозиции представляет собой способ моделирования типовой ситуации, когда любое действие, операция, функция могут быть разбиты (декомпозированы) на более простые действия, операции, функции. Другими словами, сложная бизнес-функция может быть представлена в виде совокупности элементарных функций. Представляя функции графически, в виде блоков, можно как бы заглянуть внутрь блока и детально рассмотреть ее структуру и состав.

С помощью принципа декомпозиции главная бизнес функция была разбита на 4 активные составляющие: войти в БД, выбрать сеанс, заказать билет, обработать заказ. Эта последовательность в общей сложности и представляет из себя поход в кинотеатр. Декомпозиция контекстной диаграммы функциональной модели показана на рисунке 2.3

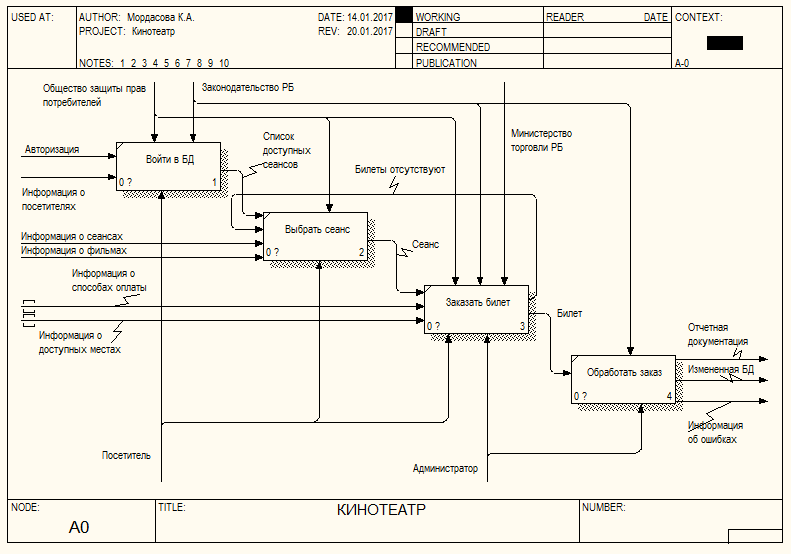


Рисунок 2.3 - Декомпозиция контекстной диаграммы

Для построения модели использовался продукт BPwin 7.3.3.1773 фирмы Computer Associates.

1. **ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ**

Процесс разработки программного средства начинается с этапа проектирования диаграмм, отражающих процесс функционирования приложения. Для проектирования диаграмм используется унифицированный язык моделирования UML. Он определяет следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма объектов, диаграмма последовательностей, диаграмма взаимодействия, диаграмма состояний, диаграмма активности, диаграмма развертывания.

### **Диаграмма вариантов использования**

Функционирование приложения можно отразить с помощью диаграммы вариантов использования, диаграммы деятельности, диаграммы последовательности и диаграммы классов.

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе.

Важно понимать, что диаграммы вариантов использования не предназначены для отображения проекта и не могут описывать внутреннее устройство системы. Диаграммы вариантов использования предназначены для упрощения взаимодействия с будущими пользователями системы, с клиентами, и особенно пригодятся для определения необходимых характеристик системы. Другими словами, диаграммы вариантов использования говорят о том, что система должна делать, не указывая сами применяемые методы.

Диаграмма вариантов использования представлена на рисунке 3.1.1.

В качестве актеров данной программы выступают: пользователь (или клиент), администратор (персонал кинотеатра имеющий доступ к программе), а также в качестве актера частично выступает банк, но он не имеет конкретную роль в работе системы, он выступает как сторонний процесс, необходимый для полной функциональности системы.

Как администратор, так и пользователь для работы с системой должен авторизоваться в базе данных, а уже потом взаимодействовать с основными функциями программы. Пользователю предоставляется достаточно малая часть функций, он может лишь посмотреть доступные фильмы, сеансы с этими фильмами и конечно заказать билет.

Вариант использования «Оформить билет» подразумевает под собой несколько последовательных действий и связано на диаграмме с ними отношением включения

Для администратора системы предоставляется больший доступ к системе, такой как редактирование добавления и удаления записей в базу данных. Так же Администратор связан с Банком через вариант использования «Информация о платежах». А он уже связан отношением включения с вариантами использования «Информация о клиенте» и «Дата поступления платежа». Это обосновано тем, что во время выполнения различных транзакций в банке, юридическое лицо, пользующееся услугами банка, получает полную отчетность о зачислении на свой счет денежных средств. А так же получает всю информацию о плательщике (посетителе кинотеатра).

Диаграмма вариантов использования показана на рисунке 3.1.



Рисунок 3.1 – Диаграмма вариантов использования

### **3.2 Диаграмма классов**

Центральное место в объектно-ориентированном программировании занимает разработка логической модели системы в виде диаграммы классов. Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

Диаграмма классов представляет собой граф, вершинами которого являются элементы типа «классификатор», связанные различными типами структурных отношений. Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи.

Существует два вида:

* статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Класс Tickets наследует свойства класса Shows, а он наследуется от классов Places и Films. В остальном же случае классы связаны элементом отношения.

Диаграмма классов представлена на рисунке 3.2.



Рисунок 3.2 – Диаграмма классов

**3.3 Диаграмма деятельности**

При моделировании поведения системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности.

Диаграмма деятельности — UML-диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому. На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности.

Компонентами диаграммы деятельности являются:

* состояния действия;
* переходы;
* дорожки;
* объекты.

Диаграмма деятельности для реализуемого программного средства начинается с начального узла, далее подключение к БД и авторизация, а затем идет разделение действий, в зависимости от того, кто подключился к базе данных администратор или простой пользователь (клиент).

Со стороны администратора идет распараллеливание действий, так как администратор может выполнять различные операции с базой данных, такие как добавление, изменение или удаления данных. Просмотр доступных данных, к примеру фильмов, сеансов, данные о посетителях, купленных билетах, доступных местах и т.д. А затем опять слияние воедино (синхронизацию) потоков управления.

Остальные действия выполняются последовательно.

Диаграмма деятельности представлена на рисунке 3.3.



Рисунок 3.3 – Диаграмма деятельности

**3.4 Диаграмма развёртывания**

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если разрабатывается программа, выполняющаяся локально на компьютере пользователя и не использующая периферийных устройств и ресурсов, то в разработке дополнительных диаграмм нет необходимости. При разработке же корпоративных приложений наличие таких диаграмм может быть крайне полезным для решения задач рационального размещения компонентов в целях эффективного использования распределенных вычислительных и коммуникационных ресурсов сети, обеспечения безопасности и других. Для представления общей конфигурации и топологии распределенной программной системы в UML предназначены диаграммы развертывания.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Разработка диаграммы развертывания, как правило, является последним этапом спецификации модели программной системы.

При разработке диаграммы развертывания преследуют следующие цели:

– определить распределение компонентов системы по ее физическим узлам;

– показать физические связи между всеми узлами реализации системы на этапе ее исполнения;

– выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Диаграмма развёртывания представлена на рисунке 3.4.



Рисунок 3.4 – Диаграмма развёртывания

**3.5 Диаграмма последовательности**

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. Во-первых, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Временной аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели в языке UML используются диаграммы последовательности.

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов, упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML.

Основными компонентами диаграммы последовательности являются:

* объекты;
* линия жизни;
* фокусы управления;
* сообщения;
* ветвления.

В данной диаграмме рассмотрена последовательность действий покупки билета со стороны пользователя. Работа системы начинается с авторизации пользователя в системе. А заканчивается получением билета.

Диаграмма последовательности представлена на рисунке 3.5.



Рисунок 3.5 – Диаграмма последовательности

### **3.6 Диаграмма коопераций**

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи - потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

Диаграмма коопераций представлена на рисунке 3.6.



Рисунок 3.6 – Диаграмма коопераций

### **Диаграмма компонентов**

Диаграмма компонентов — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

Далее проектируется логическая структура системы с помощью диаграммы компонентов. На данном этапе выделяются компоненты исходного кода, формирующие структуру системы, а также компоненты, реализующие некий набор операций, способствующий достижению целей в рамках выбранного прецедента.

Данный вид диаграмм отражает следующие аспекты проектируемой системы:

* обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними системами)
* ограничения, накладываемые на взаимодействие объектов
* события, инициирующие взаимодействия объектов.

В диаграмме представлена визуализация общей структуры исходного кода программной системы

Компонент «Cinema» получает необходимые данные от компонента «Shows», а именно показы, проходящие в кинотеатре, и генерирует список сеансов, далее именно с помощью этого компонента происходит оформление билетов и дальнейшая их реализация.

Как упоминалось ранее компонент «Shows», получает и обрабатывает данные по всем фильмам, сеансам из базы данных и обрабатывает их. К этому модулю есть доступ у персонала кинотеатра, они могут получать информацию от этого компонента и манипулировать ей.

Компонент «Employee», реализует все те действия, за которые ответственны работники кинотеатра. Подключение к БД, любые изменения в БД могут делать исключительно администраторы.

Компонент «CinemaDatabase», подразумевает под собой целостную базу данных и СУБД для работы с ней. Там храниться вся информация по фильмам, клиентам, сотрудникам, показам и др.

Компонент «Clients» - модуль, который отвечает лишь за “клиентскую” часть работы с системой. Клиенты кинотеатра получают косвенный доступ к базе данных через компоненты «Cinema» и «Shows».

Компонент «Main» - это главный модуль рабочей системы.

Диаграмма компонентов представлена на рисунке 3.7.



Рисунок 3.7 – Диаграмма компонентов

1. **ГЕНЕРАЦИЯ КОДА**

Одним из наиболее важных свойств программы IBM Rational Rose является возможность генерации программного кода на нескольких языках программирования, которая может быть использована разработчиком после построения модели.

Общая последовательность действий, которые необходимо выполнить для генерации программного кода в среде IBM Rational Rose, состоит из следующих этапов:

* проверка модели на отсутствие ошибок;
* создание компонентов для реализации классов;
* отображение классов на компоненты;
* выбор языка программирования для генерации текста программного кода;
* установка свойств генерации программного кода;
* выбор класса, компонента или пакета;
* генерация программного кода.

В качестве языка программирования был выбран язык ANSI C++.

Процесс генерации кода представлен на рисунках 4.1 – 4.4.

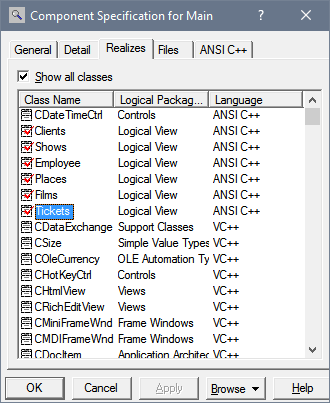


Рисунок 4.1 – Спецификация главного компонента программы

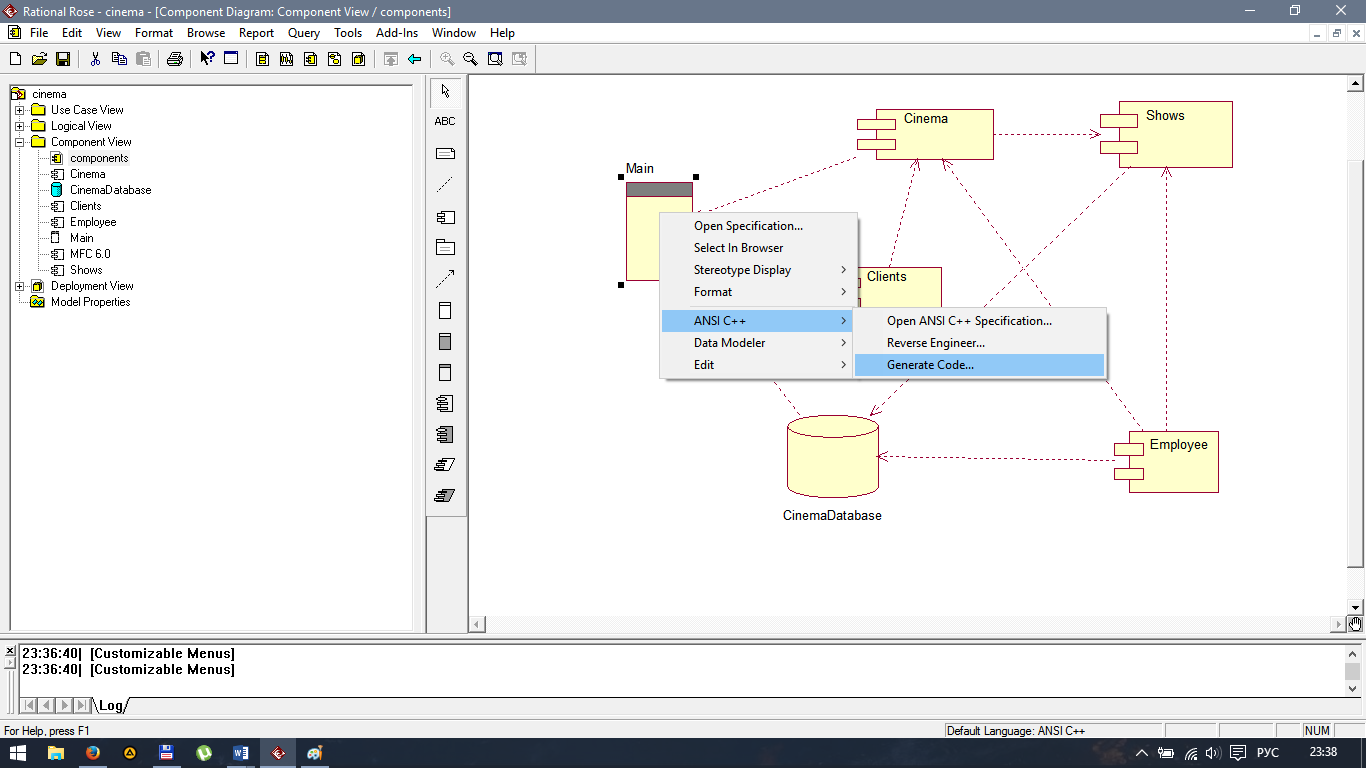


Рисунок 4.2 – Выбор языка программирования

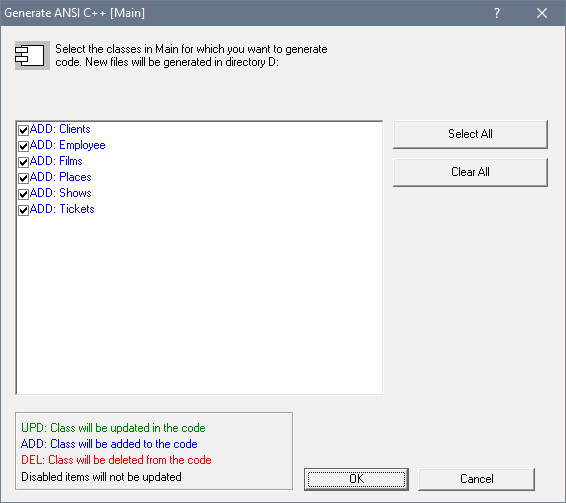


Рисунок 4.3 – Выбор классов для генерации кода

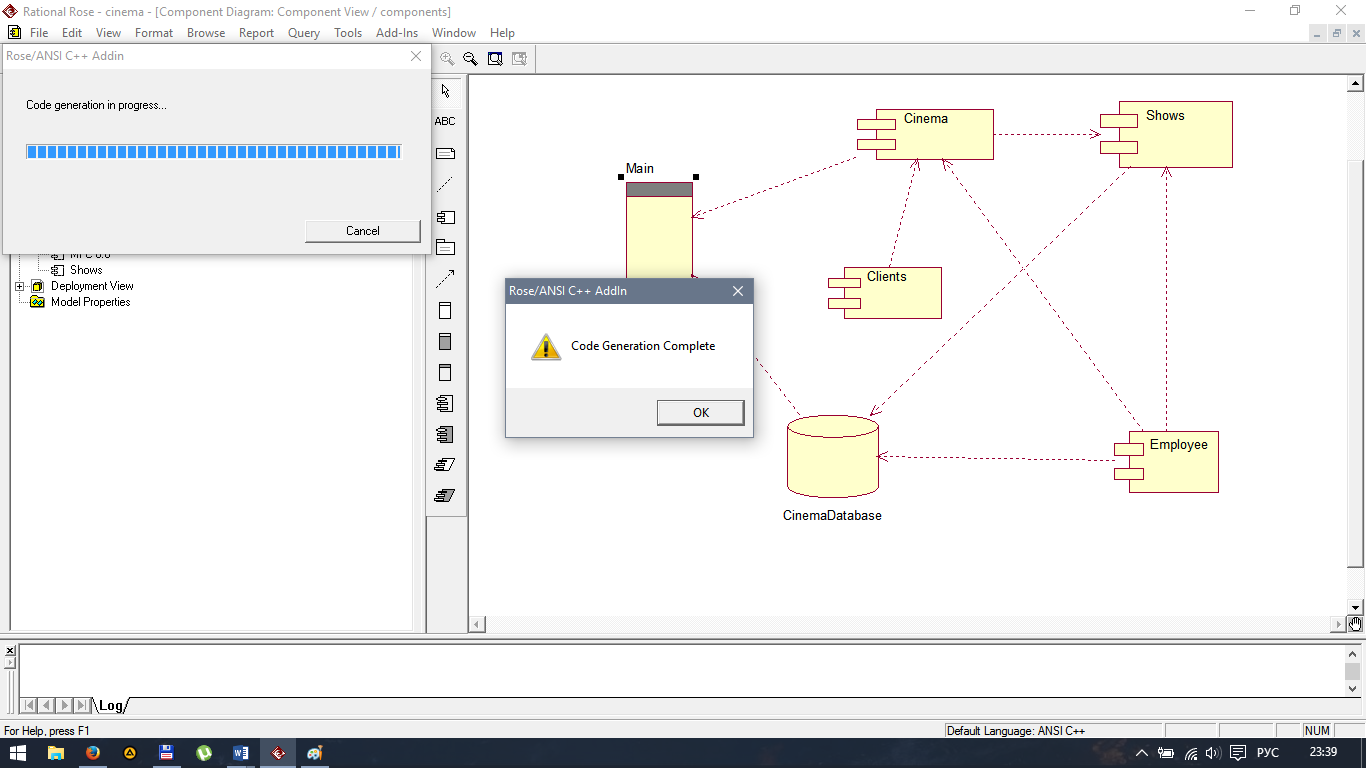


Рисунок 4.4 –Процесс генерации кода

Класс Clients:

**Clients.cpp:**

#include "Clients.h"

//##ModelId=587BD7870316

Clients::Clients()

{

}

//##ModelId=587BD7C700D5

string Clients::getshows()[]

{

}

//##ModelId=587BDA1901C3

int Clients::buyTick()

{

}

//##ModelId=587BDA410039

bool Clients::sendCheck()

{

}

**Clients.h**

#ifndef CLIENTS\_H\_HEADER\_INCLUDED\_A782AB54

#define CLIENTS\_H\_HEADER\_INCLUDED\_A782AB54

//##ModelId=587BD650009C

class Clients

{

public:

//##ModelId=587BD7870316

Clients();

// Выводит массив всех доступных показов клиента

//##ModelId=587BD7C700D5

string getshows()[];

// Оплатить показ

//##ModelId=587BDA1901C3

int buyTick();

// Отправить чек

//##ModelId=587BDA410039

bool sendCheck();

private:

//##ModelId=587BD67B00A0

string id\_client;

// Фамилия

//##ModelId=587BD6C800B9

string fname;

// Имя

//##ModelId=587BD6DA00F0

string lname;

// Данные по карточке

//##ModelId=587BD70D02C9

int card;

// id билета

//##ModelId=587BD73C0137

int id\_tick;

};

#endif /\* CLIENTS\_H\_HEADER\_INCLUDED\_A782AB54 \*/

**Employee.cpp**

#include "Employee.h"

//##ModelId=587BD8D6005C

Employee::Employee()

{

}

//##ModelId=587BD8F101A4

bool Employee::addnewadm()

{

}

//##ModelId=587BD930017B

bool Employee::GetCheck()

{

}

//##ModelId=587BD9920259

Employee::printTick()

{

}

//##ModelId=587BD9DC0112

Employee::editShows()

{

}

**Employee.h**

#ifndef EMPLOYEE\_H\_HEADER\_INCLUDED\_A782C172

#define EMPLOYEE\_H\_HEADER\_INCLUDED\_A782C172

//##ModelId=587BD81D025A

class Employee

{

public:

//##ModelId=587BD8D6005C

Employee();

// Добавляет нового пользователя административного персонала

//##ModelId=587BD8F101A4

bool addnewadm();

// Получает чек от клиента, и возвращает разрешение на оформление билета

//##ModelId=587BD930017B

bool GetCheck();

// РАспечатать билет

//##ModelId=587BD9920259

printTick();

// Позволяет добавлять новые показы и удалять старые

//##ModelId=587BD9DC0112

editShows();

private:

//##ModelId=587BD8570115

int id\_emp;

// Фамилия сотрудника

//##ModelId=587BD8810125

string fname;

// Имя сотрудника

//##ModelId=587BD89B00EB

string lname;

// Должность

//##ModelId=587BD8B00113

string position;

};

#endif /\* EMPLOYEE\_H\_HEADER\_INCLUDED\_A782C172 \*/

**Films.cpp**

#include "Films.h"

//##ModelId=587BE09002E3

Films::Films()

{

}

//##ModelId=587BE09C01BA

bool Films::addNewFilm()

{

}

**Films.h**

#ifndef FILMS\_H\_HEADER\_INCLUDED\_A782FBC9

#define FILMS\_H\_HEADER\_INCLUDED\_A782FBC9

#include "Places.h"

//##ModelId=587BDF5A0183

class Films : public Places

{

public:

//##ModelId=587BE09002E3

Films();

//##ModelId=587BE09C01BA

bool addNewFilm();

private:

// id фильма

//##ModelId=587BDF6D0296

int id\_film;

// Название фильма

//##ModelId=587BDF6E03B5

string movieTitle;

// Жанр фильма

//##ModelId=587BDF70003D

string genre;

// Страна производства

//##ModelId=587BDF7501FE

string country;

// Продолжительность фильма

//##ModelId=587BDF7601FE

int time;

// Дата начала проката

//##ModelId=587BDF78022D

int rentalDate;

};

#endif /\* FILMS\_H\_HEADER\_INCLUDED\_A782FBC9 \*/

**Places.cpp**

#include "Places.h"

//##ModelId=587BDE330391

Places::Places()

{

}

//##ModelId=587BDED701F6

int Places::getEmptyPlace()[]

{

}

**Places.h**

#ifndef PLACES\_H\_HEADER\_INCLUDED\_A782DCAF

#define PLACES\_H\_HEADER\_INCLUDED\_A782DCAF

//##ModelId=587BDDED038D

class Places

{

public:

//##ModelId=587BDE330391

Places();

//##ModelId=587BDED701F6

int getEmptyPlace()[];

private:

// Номер зала

//##ModelId=587BDE3B0030

int hallRoom;

// ряд

//##ModelId=587BDE77024F

int row;

// Место

//##ModelId=587BDE8F01BF

int place;

// Текущее состояние (занято(true)/свободно(false))

//##ModelId=587BDF2302AD

bool state;

};

#endif /\* PLACES\_H\_HEADER\_INCLUDED\_A782DCAF \*/

**Shows.cpp**

#include "Shows.h"

//##ModelId=587BE10301DB

Shows::Shows()

{

}

**Shows.h**

#ifndef SHOWS1\_H\_HEADER\_INCLUDED\_A782A3CB

#define SHOWS1\_H\_HEADER\_INCLUDED\_A782A3CB

#include "Films.h"

#include "Places.h"

//##ModelId=587BE0B8027D

class Shows : public Films, public Places

{

public:

//##ModelId=587BE10301DB

Shows();

private:

//##ModelId=587BE1090212

int id\_show;

// Фильм

//##ModelId=587BE11C036A

string film;

// Дата показа

//##ModelId=587BE13D0042

int Date;

// Время показа

//##ModelId=587BE15D01D2

int Time;

//##ModelId=587BE16C02DA

int hallRoom;

};

#endif /\* SHOWS1\_H\_HEADER\_INCLUDED\_A782A3CB \*/

**Tickets.cpp**

#include "Tickets.h"

//##ModelId=587BDCF803B3

Tickets::Tickets()

{

}

**Tickets.h**

#ifndef TICKETS\_H\_HEADER\_INCLUDED\_A782E347

#define TICKETS\_H\_HEADER\_INCLUDED\_A782E347

#include "Shows.h"

//##ModelId=587BDCEC0096

class Tickets : public Shows

{

public:

//##ModelId=587BDCF803B3

Tickets();

private:

// id билета

//##ModelId=587BDD040172

int id\_tick;

// Номер ряда

//##ModelId=587BDD34033B

int row;

// Место в ряду

//##ModelId=587BDD580022

int place;

// Название фильма

//##ModelId=587BDD9401F9

string film;

// Дата и время показа

//##ModelId=587BDDB90168

int showDate;

};

#endif /\* TICKETS\_H\_HEADER\_INCLUDED\_A782E347 \*/

**ЗАКЛЮЧЕНИЕ**

Предметной областью является программное средство «Информационная система кинотеатра». Разработанное программное средство оптимизирует и автоматизирует работу кинотеатра.

Контрольная работа состоит из следующих разделов:

1. Постановка задачи. В данном разделе была поставлена задача по разработке программного средства «Информационная система кинотеатра».

2. Описание основного процесса предметной области с использованием стандарта IDEF0.

3. Описание моделей информационной системы с использованием языка UML.

4 Генерация кода на основе моделей Rational Rose.

В ходе написания контрольной работы были закреплены теоретические знания по таким разделам как составление функциональной модели (IDEF0), проектирование системы с помощью [унифицированного языка моделирования](http://edu.dvgups.ru/METDOC/GDTRAN/YAT/ITIS/PROEK_INF_SIS/METOD/UMK_DO/frame/UMK_DO/M6/L11.htm#11_1) (UML).

Контрольная работа выполнялась в соответствии с методическими требованиями и указаниями.

**СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ**

1. ГОСТ 2.105-95 ЕСКД. Введ. 1996–07–01. – М.: Изд-во стандартов, 1996.
2. ГОСТ 19.104–78 – Единая система программной документации. Основные надписи.
3. Маклаков С.В. BPwin и ERwin: CASE - средства для разработки информационных систем.
4. Федотов Д.Э., Семенов Ю.Д., Чижик К.Н. Практикум для высших учебных заведений. CASE-технологии.-157с.
5. Интернет обучение [Электронный ресурс] / Котляр Д.С. Использование CASE-средства ERwin для автоматизации проектирования и разработки базы данных – Режим доступа: http://royallib.com/read/ bezopasnost – Дата доступа: 15.12.2015.
6. Интернет обучение [Электронный ресурс] / Информационные системы и технологии – Режим доступа: <http://www.narfu.ru> – Дата доступа: 19.12.2016.
7. Интернет обучение [Электронный ресурс] / Нормализация структурны данных – Режим доступа: http://infostart.ru/public/269803/ – Дата доступа: 15.01.2017.
8. Интернет обучение [Электронный ресурс] / Программа компьютерного моделирования BpWin – Режим доступа: http://bourabai.kz/cm/bpwin.htm– Дата доступа: 11.11.2016.
9. Леонков, А. В. Самоучитель UML / А. В. Леонков. 2-е изд. – СПб.: БХВ - Петербург, 2007. – 596 с.
10. Трофимов, С.А. CASE-технологии. Практическая работа в Rational Rose / С. А. Трофимов - Бином-Пресс 2002 . – 288 с.
11. Хабрахабр, UML – диаграмма вариантов использования (use case diagram) [Электронный ресурс]. – 2013. – Режим доступа: http://habrahabr.ru/post/47940/ . – Дата доступа: 15.01.2017.
12. Сайтмонитор, Язык UML. Руководство пользователя [Электронный ресурс]. – 2013. – Режим доступа: http://sitemonitor.ru/doc/UML\_HTM/ . – Дата доступа: 15.01.2017.