Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Системы автоматики, автоматизированное управление и проектирование

УТВЕРЖДАЮ
Заведующий кафедрой
_____С. В. Ченцов
«____» _____2017г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПРОИЗВОДСТВЕННОГО ПЛАНИРОВАНИЯ

Направление 27.04.04 Управление в технических системах
Магистерская программа 27.04.04.01 Интегрированные системы управления производством

Научный руководитель	_____	_____.2017 г.	доцент, канд. техн. наук Д. В. Капулин
Выпускник	_____	_____.2017 г.	Д. И. Винниченко
Рецензент	_____	_____.2017 г.	зав. каф., канд. техн. наук А. С. Кузнецов
Нормоконтролер	_____	_____.2017 г.	Т. А. Грудинова

Красноярск 2017

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка программного обеспечения автоматизированной системы производственного планирования» содержит 69 страниц текстового документа, 38 рисунков, 1 таблицу, 29 использованных источников, 4 приложения.

СЕТЕВОЙ ГРАФИК, СЕТЕВОЕ ПЛАНИРОВАНИЕ, ОБЪЕМНО-КАЛЕНДАРНЫЙ ПЛАН, КАЛЕНДАРНЫЙ ГРАФИК, ASP.NET Core 1.0, Java Script, MVC, DHTMLX.

Целью данной выпускной квалификационной работы является разработка структур и моделей оперативного управления производством, позволяющих максимально точно визуализировать сетевую модель с использованием диаграммы Ганта.

В процессе диссертационного исследования реализован алгоритм сетевого производственного планирования с использованием CASE-инструментария, осуществлен переход от абстракций и алгоритмов предметной области, предоставленных аналитиком, к терминам объектно-ориентированной парадигмы; получены модели на языке UML, осуществлена их трансляция на язык C# с поддержкой механизма RTE для непрерывной синхронизации кода и модели.

Получена библиотека имплементации предметной области, пригодная для использования в других проектах отдела АСУП предприятия. С использованием технологии ASP.NET MVC разработано и развернуто на сервере приложений предприятия экспериментальное web-приложение для построения календарного графика, работающее с массивами отдела АСУП.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Применение системы сетевого планирования способствует разработке оптимального варианта стратегического плана развития предприятия, который служит основой оперативного управления комплексом работ в ходе его осуществления. Основным плановым документом в этой системе является сетевой график, представляющий собой информационно-динамическую модель, в которой отражаются все логические взаимосвязи и результаты выполняемых работ, необходимых для достижения конечной цели стратегического планирования. В сетевом графике с необходимой степенью детализации изображается, какие работы, в какой последовательности и за какое время предстоит выполнить, чтобы обеспечить окончание всех видов деятельности не позже заданного или планируемого периода.

При сетевом планировании производства:

1) видна цепочка работ, от которых зависит своевременное выполнение проекта;

2) используются простые математические зависимости;

3) выявляются резервы, которые можно использовать внутри проекта и, следовательно, сократить длительность и стоимость.

Каким бы совершенным не был производственный процесс, на предприятии всегда найдутся внутрипроизводственные резервы. С течением времени в силу появления новых достижений научно-технического прогресса величина этих резервов будет возрастать. В настоящий момент на научно-производственном предприятии «Радиосвязь» процессом построения календарных графиков занимается производственный отдел, его работники используют массивы данных, предоставляемые отделом АСУП, и осуществляют обсчет в ручном режиме, поэтому актуальной задачей является автоматизация этого процесса.

Цель диссертационной работы состоит в разработке структур и моделей оперативного управления производством АО «НПП «Радиосвязь», позволяющих максимально точно визуализировать сетевую модель с использованием диаграммы Ганта.

В работе формулируются и решаются задачи, необходимые для достижения поставленной цели:

– анализ существующих технологий для разработки веб-приложения;

– переход с платформы ASP.NET 4.5 на ASP.NET Core 1.0;

– логическое структурирование данных для системы оперативного управления производством;

– разработка программной архитектуры;

– расширение функционала сетевой модели;

– внедрение подсистемы на предприятие.

Методы исследования. В диссертации использованы методы системного анализа и объектно-ориентированного подхода. Проектирование алгоритмического и программного обеспечения выполнено с использованием средств языка UML. При реализации использовалась среда web-разработки ASP.NET MVC, язык программирования C#, JavaScript.

Научная новизна результатов исследования заключается в предложенной архитектуре и разработанном программном обеспечении, позволяющие реализовать динамическую корректировку производственного заказа в условиях изменения конструкторской и технологической документации.

Магистерская диссертация состоит из введения, трёх разделов, заключения, списка использованных источников, списка сокращений и приложений.

Во введении обосновывается актуальность темы, определяется цель научно – исследовательской работы и перечень решаемых задач, излагается основная идея диссертации, перечисляются основные методы проведенных исследований.

В первом разделе рассматриваются основные принципы сетевого планирования. Дается обзор основных компонентов.

Во втором разделе производится сравнительный анализ нескольких сред для разработки веб-приложения. Далее происходит обзор шаблонов проектирования, выбор наиболее подходящего и обоснование.

В третьем разделе происходит выявление сущностей и построение диаграммы классов предметной области, а так же описываются основные расчетные операции в виде диаграмм UML. Следующим этапом разрабатывается MVC приложение.

В ходе исследования были реализованы требования к автоматизированной системе планирования. Описаны основные механизмы для реализации системы. Представлено визуальное отображение календарного графика.

1 Анализ современных подходов к автоматизации процессов производственного планирования

1.1 Анализ процессов производственного планирования

Своевременное выполнение предприятием производственного плана во многом зависит от результата и согласованности действий всех его цехов и отделов. Для своевременного обеспечения материальными и производственными ресурсами, а также для решения поставленной задачи требуется обозначить плановые задания по производству готовой продукции, головных блоков и деталей для цехов, участков и рабочих мест на короткие промежутки времени (час, день, неделя, месяц, полугодие, год). Технико-экономическое планирование не обеспечивает решения таких задач, поэтому для таких целей применяются системы оперативно-календарного планирования.

Оперативно-календарное планирование (ОКП) позволяет организовать слаженную работу всех цехов и отделов предприятия для обеспечения своевременного изготовления продукции согласно поставленным в контрактах объемам, номенклатуре и срокам поступления при наиболее полном использовании всех ресурсов. Использование ОКП обеспечивает непрерывность производственного процесса, ритмичность производства и закрепление взаимосвязи между его отдельными элементами.

Оперативно-календарное планирование (ОКП) – детализация годового (квартального, месячного) производственного плана завода по срокам запуска-выпуска изделий, блоков и деталей по всем технологическим операциям производственного цикла. Существует два вида ОКП:

– внутрицеховое оперативное планирование: организация выполнения производственных заданий, сформированных для определенного цеха; разработка календарных планов-графиков на короткие промежутки времени (месяц, смену, час). Предназначением внутрицехового планирования

являются также оперативное регулирование, учет и контроль выполнения планов;

– межцеховое оперативное планирование: обеспечение слаженной работы всех основных цехов по выполнению общезаводской производственной программы; разработка взаимосвязанных производственных процессов, следующих из производственной программы завода. Календарно-плановые нормативы производственных процессов и сквозные графики являются визуальными средствами этого вида планирования [1].

Эффективное использование всех ресурсов предприятия обеспечивает выполнение подразделениями одинаковых объемов работ за равные или кратные промежутки времени. Ритмичная работа предприятия – это, прежде всего, показатель высокого уровня производства и показатель его конкурентоспособности. Равномерность производства проявляется в том, что:

– соблюдаются требования потребителей по качеству, номенклатуре, объему и срокам поставки продукции;

– обеспечивается рост производительности труда за счет исключения издержек на производстве: простоев рабочих мест, сокращения технологического отхода, постоянная загрузка станков;

– обеспечивается снижение себестоимости продукции за счет уменьшения труднореализуемых или браковочных деталей, расходование денежных средств на оплату сверхурочных и внеплановых работ.

Организованная система ОКП позволяет максимально сократить перерывы в движении средств производства через последовательные стадии технологического процесса, что приводит к уменьшению незавершенного производства, повышению плотности производственного цикла, а также к ускорению оборачиваемости оборотных средств. Важной задачей ОКП является обеспечение равномерной загрузки оборудования, что приводит к увеличению прибыли.

Следует отметить, что слаженная работа предприятия обеспечивается не только устойчивой системой ОКП, но и некоторым набором правил, к важнейшим из которых следует отнести:

– обеспечение рационального использования мощностей (пропускной способности) производственных подразделений;

– повышение качества и актуальности технической и технологической подготовки производства;

– внедрение поточных методов организации производства;

– последовательное осуществление специализации предприятия и его производственных подразделений;

– точная организация материально-технического обеспечения предприятия [1].

Одним из главных требований, предъявляемых к организации ОКП, является гибкость. Система ОКП должна обеспечивать быструю реакцию на технологические и номенклатурные изменения в производственных системах и для обеспечения надлежащих условий выполнения плановых заданий в соответствии с установленными требованиями устранять появляющиеся отклонения в производственном процессе.

Такой подход возможно реализовать только при использовании информационных технологий. Современное оборудованием можно запрограммировать на автоматизированное управление ходом производства и сигнализировать при возникновении отклонений. Тем самым работник может сосредоточить все свое внимание на решение нестандартных, аварийных ситуаций.

Основными функциями оперативно-календарного планирования являются разработка и планирование:

– номенклатурно-календарных планов запуска (выпуска) продукции предприятия его основными подразделениями;

– календарно-плановых нормативов движения производства;

– расчетов загрузки оборудования и производственных площадей.

Более того, службы, разрабатывающие ОКП, обязаны контролировать выполнения плана производственными подразделениями, вести оперативный учет и оперативное регулирование производства (при возникновении отклонений реализовать меры по их устранению). Обязанностями служб оперативно-календарного планирования являются не только проверка соблюдения графика технической подготовки производства новой продукции, но также контроль за ходом производства в цехах.

Отличительной особенностью оперативно-календарного планирования является своевременным снабжением каждого рабочего места необходимыми материалами и комплектующими, инструментом. Следовательно, функции оперативно-календарного планирования хорошо сочетаются с функциями оперативного управления производством.

Диспетчирование (организация выполнения плановых заданий) является также немаловажной функцией. Оно представляет собой непрерывное наблюдение и контроль за ходом производства на основе получения оперативной информации о фактическом выполнении цеховых планов-графиков, сменно-суточных заданий и обо всех отклонениях от плана. Получение оперативной информации происходит на основе средств связи, вычислительной техники и компьютеров, которые обеспечивают автоматизацию получения, переработки и передачи информации. Такой цикл повышает качество и обоснованность оперативных управленческих решений, которые позволяют в кратчайшие сроки корректировать графики движения средств производства по разным производственным процессам и исключить любые отклонения от плана [2].

1.2 Трёхуровневая архитектура

Трёхуровневая архитектура (рисунок 1.1) – архитектурная модель компонентов программного комплекса, предполагающая наличие в нем трёх компонентов: клиента, сервера приложений (к которому подключено

клиентское приложение) и сервера баз данных (с которым работает сервер приложений).

Рисунок 1.1 – Трехуровневая схема

Клиент (слой клиента) – это интерфейсный (обычно графический) компонент комплекса, предоставляемый конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных (по требованиям безопасности и масштабируемости), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надёжности). На этот уровень обычно выносится только простейшая бизнес-логика: интерфейс авторизации и аутентификации, проверка входных значений на допустимость и соответствие формату, простые операции с данными (сортировка, группировка, подсчёт итоговых значений), уже загруженными на автоматизированном рабочем месте (АРМ) клиента [3].

Сервер баз данных (слой данных) обеспечивает обработку, хранение и передачу данных, который выносится на отдельный уровень, реализуется, как правило, средствами систем управления базами данных(СУБД),

подключение к этому компоненту обеспечивается только с уровня сервера приложений посредством специализированных драйверов.

Сервер приложений (связующий слой) располагается на втором уровне, на нём содержится большая часть бизнес-логики приложения. Вне этого слоя содержатся только фрагменты, экспортируемые на клиента (рабочие места), а также элементы серверной логики, которые находятся в базе данных (триггеры, функции и хранимые процедуры). Реализация данного компонента обеспечивается связующим программным обеспечением. Серверы приложений проектируются таким образом, чтобы добавление к ним дополнительных экземпляров обеспечивало горизонтальное масштабирование производительности программного комплекса и не требовало внесения изменений в исходный код приложения.

В простейших конфигурациях все компоненты или часть из них могут быть совмещены на одном вычислительном узле. В продуктивных конфигурациях, как правило, используется выделенный вычислительный узел для сервера баз данных или кластер серверов баз данных, для серверов приложений – выделенная группа вычислительных узлов, к которым непосредственно подключаются клиенты (терминалы).

По сравнению с клиент-серверной или файл-серверной архитектурой трёхуровневая архитектура обеспечивает, как правило, большую масштабируемость (за счёт горизонтальной масштабируемости сервера приложений и мультиплексирования соединений, большую конфигурируемость (за счёт изолированности уровней друг от друга), более широкие возможности по обеспечению безопасности и отказоустойчивости. Кроме того, в сравнении с клиент-серверными приложениями, использующими прямые подключения к серверам баз данных, снижаются требования к скорости и стабильности каналов связи между клиентом и серверной частью. Реализация приложений, доступных из веб-браузера или из тонкого клиента, как правило, подразумевает развёртывание программного комплекса в трёхуровневой архитектуре. При этом обычно

разработка приложений для трёхуровневых программных комплексов сложнее, чем для клиент–серверных приложений, также наличие дополнительного связующего программного обеспечения может налагать дополнительные издержки в администрировании таких комплексов.

1.3 Данные (база данных SQL)

Формальный непроцедурный язык программирования SQL применяется для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД). Язык SQL основывается на исчислении кортежей. В SQL Server существует два типа ролей уровня базы данных: предопределенные роли базы данных, стандартные для базы данных, и гибкие роли базы данных, которые может создать пользователь [4].

Предопределенные роли базы данных задаются на уровне базы данных и предусмотрены в каждой базе данных. Члены ролей базы данных db_owner и db_securityadmin могут управлять членством в предопределенных ролях базы данных. Но только члены роли базы данных db_owner могут добавлять членов в предопределенную роль базы данных db_owner. Кроме того, в базе данных msdb имеются специальные предопределенные роли базы данных.

В роли уровня базы данных можно добавить любую учетную запись базы данных и другие роли SQL Server. Каждый член предопределенной роли базы данных может добавлять другие имена входа к той же роли.

Реляционная база данных – база данных, основанная на реляционной модели данных. Реляционная СУБД (РСУБД), реже – система управления реляционными базами данных (СУРБД) – СУБД, управляющая реляционными базами данных. Практически все разработчики современных приложений, предусматривающих связь с системами баз данных, ориентируются на реляционные СУБД [1]. Реляционные СУБД используются

в абсолютном большинстве крупных проектов по разработке информационных систем, только около 7% составляют проекты, в которых используются СУБД не реляционного типа [5].

1.4 Бизнес-логика (сервер приложений IIS)

Бизнес-логика – совокупность правил, принципов, зависимостей поведения объектов предметной области; реализация правил и ограничений автоматизируемых операций. В рассматриваемом случае, предметной областью являются процессы производственного планирования приборостроительного предприятия, отличающиеся мелкосерийностью и высокой частотой смены производственных заказов.

В фазе бизнес–моделирования и разработки требований бизнес-логика может описываться в виде:

- текста;
- концептуальных аналитических моделей предметной области;
- бизнес-правил;
- разнообразных алгоритмов;
- диаграмм деятельности;
- графов и диаграмм перехода состояний;
- моделей бизнес-процессов.

В фазе анализа и проектирования системы бизнес-логика воплощается в различных диаграммах языка UML или ему подобных. В фазе программирования бизнес-логика воплощается в коде классов и их методов, в случае использования объектно–ориентированных языков программирования, или процедур и функций, в случае применения процедурных языков [6].

Стоит отметить, что бизнес-логикой также могут называться программные модули, её реализующие, и уровень автоматизированной системы, на котором эти модули находятся. В многоуровневых

(многослойных) информационных системах этот уровень взаимодействует с нижележащим уровнем инфраструктурных сервисов, например, интерфейсом доступа к базе данных или файловой системе и вышележащим уровнем сервисов приложения, который уже, в свою очередь, взаимодействует с уровнем пользовательского интерфейса или внешними системами.

IIS (Internet Information Services) – проприетарный набор серверов для нескольких служб Интернета от компании Майкрософт. IIS распространяется с операционными системами семейства Windows NT. Основным компонентом IIS является веб-сервер, который позволяет размещать в Интернете сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP.

Основным компонентом IIS является веб-сервер – служба WWW (называемая также W3SVC), которая предоставляет клиентам доступ к сайтам по протоколам HTTP и, если произведена настройка, HTTPS.

Один сервер IIS может обслуживать несколько сайтов (IIS 6.0 и выше). Каждый сайт имеет следующие атрибуты:

– IP-адрес сайта;

– TCP-порт, на котором служба WWW ожидает подключений к данному сайту;

– заголовок узла (Host header name) – значение заголовка Host запроса HTTP указывающее обычно DNS – имя сайта.

Таким образом, например, один сервер с одним IP-адресом может обслуживать на одном TCP-порту несколько сайтов. Для этого необходимо создать несколько DNS-записей, указывающих на IP-адрес сервера, и различать сайты по заголовкам узла.

Сервер приложений – программная платформа (Фреймворк), предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов,

доступных разработчику программного обеспечения через API (Интерфейс прикладного программирования), который определен самой платформой [6].

Преимущества серверов приложений заключаются в обеспечении целостности данных и кода, централизации настройки и управления, высоком уровне безопасности и поддержке транзакций. Так, выделяя бизнес логику на отдельный сервер, или на небольшое количество серверов, можно гарантировать обновления и улучшения приложений для всех пользователей. Отсутствует риск, что старая версия приложения получит доступ к данным или сможет их изменить старым несовместимым образом. Изменения в настройках приложения, таких как изменение сервера базы данных или системных настроек, могут производиться централизованно. Сервер приложений действует как центральная точка, используя которую, поставщики сервисов могут управлять доступом к данным и частям самих приложений, что считается преимуществом защиты. Её наличие позволяет переместить ответственность за аутентификацию с потенциально небезопасного уровня клиента на уровень сервера приложений, при этом дополнительно скрывая уровень базы данных.

Транзакция представляет собой единицу активности, во время которой большое число изменений ресурсов (в одном или различных источниках) может быть выполнено атомарно (как неделимая единица работы). Конечные пользователи при этом могут выиграть от стандартизованного поведения системы, от уменьшения времени на разработку и от снижения стоимости. В то время как сервер приложений выполняет массу нужного генерирования кода, разработчики могут сфокусироваться на бизнес-логике.

1.5 Выводы по главе 1

Эффективным методом повышения производительности труда является применение системы сетевого планирования, которая способствует разработке оптимального варианта стратегического плана развития предприятия, который служит основой оперативного управления комплексом

работ в ходе его осуществления. Основным плановым документом в этой системе является сетевой график, или просто сеть, представляющий информационно-динамическую модель, в которой отражаются все логические взаимосвязи и результаты выполняемых работ, необходимых для достижения конечной цели стратегического планирования.

Все системы календарного планирования представлены с клиентской стороны, поэтому в классической архитектуре типа «клиент-сервер» приходится распределять три основные части приложения по двум физическим модулям. Обычно ПО хранения данных располагается на сервере (например, сервере базы данных), интерфейс с пользователем – на стороне клиента, а вот обработку данных приходится распределять между клиентской и серверной частями.

2 Анализ требований информационной системы предприятия

Информационная система предприятия представлена «слоями», где каждый слой обеспечивает функционирование следующих перед ним слоев:

1) исполнительный слой – набор задач, решение которых обеспечивает эффективную работу предприятия;

2) прикладной слой – набор прикладного программного обеспечение, непосредственно применяемого для обработки деловой информации;

3) системный логический слой – системное программное обеспечение, обеспечивающее функционирование физического слоя, механизмы обмена информацией, разграничение прав доступа, и т. п.;

4) физический слой – оборудование, кабельные сети, каналы передачи данных [7].

Для слоев 2, 3 и 4 возможно применение готовых типовых проектов, с изменениями под конкретные требования заказчика (локальная сеть предприятия).

Реализация системы обычно начинается с этапа проектирования исполнительного слоя, но для этого на предприятии должна быть создана модель документооборота и определены функции каждого исполнителя в этой модели.

Обычно модель документооборота тесно связана со структурой предприятия. На основе модели документооборота проектируется исполнительный слой. Определить, что должно входить в исполнительный слой системы – это и есть основная задачи проектирования исполнительного слоя. Исполнительный слой может формироваться на предприятии своими силами, возможно с привлечением сторонних специалистов.

Остальные слои проектируются последовательно друг за другом, на основе исходных данных предыдущего слоя. И если на предприятии нет опыта проектирования в данной области, то для экономии средств и времени

имеет смысл обратиться к сторонним специализированным организациям, зарекомендовавшим себя в проведении подобных работ.

2.1 Анализ информационной структуры предприятия

В настоящее время для успешного развития предприятия необходимо организовать единый непрерывный цикл деятельности, который позволит учесть любые коррективы на всех этапах жизненного цикла продукции, охватывающего все подразделения и службы организации.

Предприятие АО «НПП «Радиосвязь» учитывает эти особенности, тем самым, организует целостность данных, для которых существуют общие требования по хранению, обработке, обновлению, распространению и передачи информации. Такая совокупность информационных средств позволяет осуществить электронное взаимодействие между всеми участниками процесса подготовки производства изделия на основе соответствующих информационных технологий. Наряду с этим единое информационное пространство (ЕИП) будет хранить в себе актуальную единожды созданную информацию с учетом исключения дублирования и перекодировки в процессе обмена.

Информационной составляющей ЕИП являются не только данные, но также и применяемые программы. В таком ключе часть систем информационного пространства реализуется в виде приложений, обращение к которым возможно из других систем данного ЕИП. Например, взаимодействуют две информационные системы, одна из них будет выступать в роли клиента, а другая в роли сервера. Та система, которая будет клиентом, может пользоваться сервисами второй и получать уже готовые обработанные данные для дальнейшей обработки компонентами первой ИС. Такой подход позволяет исключить дублирование приложений в рамках одного пространства и добиться оптимизированного использования информационных ресурсов.

На предприятии организация информационной инфраструктуры реализована по трехуровневой схеме. Планирование и управление производственными процессами осуществляется силами отдела АСУП (автоматизированная система управления предприятием). Организуются АРМ на уровне цехов и отделов. Операторы на АРМ работают с внутренними Web-приложениями предприятия, реализующих бизнес-логику. Такие приложения разрабатываются на языке C# под архитектуру ASP.NET Web Forms и ASP.NET MVC и размещаются на сервере приложений IIS. При этом часть логики и расчетов, которые относятся к визуализации, реализуются на языке JavaScript с применением библиотеки JQuery. Приложения обращаются к серверу баз данных с использованием языка T-SQL для получения информации о производстве.

2.2 Анализ и сравнение существующих технологий для разработки веб-приложения

На сегодняшний день у разработчиков веб-приложений есть богатый выбор относительно того, какой язык (или технологию) использовать для создания приложения. Вариантов множество: Perl, PHP, ASP, ASP.Net, JSP, Coldfusion и другие. Самыми распространенными (с большим отрывом от остальных) сегодня являются PHP и ASP.Net.

2.2.1 Сравнительный анализ PHP и ASP.Net

PHP (Hypertext Preprocessor) – язык для написания серверных сценариев (скриптов). Интерпретатор языка бесплатен, с открытым исходным кодом, созданы версии для различных веб-серверов – прежде всего, для Apache и Internet Information Services. В последнее время этим продуктом заинтересовалась и Microsoft – началось тесное сотрудничество с фирмой Zend, в результате которого появилась встроенная поддержка PHP в

Internet Information Services 7-й версии (поставляется с Windows Server 2008) – ранее PHP можно было использовать только как внешнее расширение.

ASP.Net (Active Server Pages .Net) – это многокомпонентная технология, не являющаяся языком программирования. Средство разработки ASP.Net и принцип работы ASP.Net-приложения существенно отличается от предыдущей версии – ASP. Разницу рассмотрим далее [8].

PHP является языком web-программирования, который позволяет динамически выводить HTML-разметку и иную информацию (например, заголовки для Cookie и др.). Например:

```
<?php
print “<p>Диаграмма</p>”;
?>
```

PHP-код можно вставлять в стандартную статическую HTML-разметку:

```
<body>
    <table>
        <tr><td>
            <?php
            print “Диаграмма”;
            ?>
        </td></tr>
    </table>
</body>
```

Характерным отличием для ASP.Net Webforms относительно PHP является иная реализация принципа работы – каждая страница состоит из трех файлов: файла с HTML-разметкой, ASP.Net-контролами (файл с расширением Aspx) и файла серверной логики (расширения cs). Общая идея подхода заключается в отделении внешнего вида страницы от логики: файл разметки будет содержать только разметку (описывать внешний вид), а файл логики – только программный код. Возможность добавлять программный код в файл разметки и в файле бизнес-логики формировать HTML-разметку

не запрещается, но это уже будет подход, аналогичный PHP. Пример файла с разметкой:

```
<%@ PageLanguage="C#"AutoEventWireup="true" CodeFile="DefaultPage.aspx
.cs" Inherits="TestWebApplication._Default" %>
DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="https://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
head>
<body>
    <table>
        <tr><td>
        <asp:Label ID="LblClient" runat="server" Text="">asp:Label><br/>
        <asp:Button ID="BtnPress" runat="server" Text="Нажать"
        OnClick=" BtPress_Click" />
        </td></tr>
    </table>
</body>
</html>
```

В представленной выше разметки страницы есть теги (серверные контролы ASP.Net), которые не характерны для HTML-разметки. Например, такие теги как asp:Button, asp:TextBox будут преобразованы в <input type="text" />. Серверным контролам можно задать состояние: размер, добавить JavaScript и т.д.

Соответствующий файл с логикой имеет вид:

```
namespace TestWebApplication;
{
    public partial class _Default : System.Web.UI.Page
    {
```

```
        protected void Page_Load(object sender, EventArgs e)
            {
            }
        protected void BtnPress_Click(object sender, EventArgs e)
            {
            LblClient.Text ="Построить";
            }
    }
}
```

Все созданные на странице серверные контролы доступны в файле с бизнес-логикой по назначенному им идентификатору. К примеру, выше программно осуществляется доступ к элементу LblClient и задается текст, который он будет отображать. Действия по изменения состояния объекта осуществляются на этапе обработки страницы сервером приложений (IIS), далее у пользователя в браузер отрисовывается готовая HTML-разметка, не имеющая серверных элементов.

Даже по объему приведенных примерных текстов для каждой из технологий видно, что принципы работы PHP проще, нежели ASP.Net. Это является одной из главных причин широкого распространения PHP.

В отличие от PHP, ASP.Net – попытка применить при разработке Web-приложений те же принципы, что и при разработке Windows-приложений. Протокол HTTP – протокол, позволяющий взаимодействовать приложениям длительное время между собой по принципу вопрос-ответ. Так как Windows-приложения работают непрерывно необходимо применение множества абстракций, призваных скрыть «сеансовость» протокола HTTP. Достоинством их является то, что значительно ускоряется процесс разработки веб-приложения [8].

Сам по себе PHP выступает в роли Си-подобного языка программирования без какой-либо типизация переменных, но с поддержкой объектно-ориентированного подхода.

При создании веб-приложений ASP.Net можно использовать любой язык из платформы .Net – C# или Visual Basic.Net. Доступна вся функциональность библиотеки .Net Framework, что значительно понижает преимущество PHP.

Во-первых, по возможностям PHP отстает от DOT Net. К примеру в PHP5 была значительно улучшена поддержка ООП, но она не соответствует .Net, где языки программирования такие как C# и VB.Net являются полностью объектными. В объектно-ориентированных языках невозможно создать переменную – только атрибут класса, нельзя создать функцию – только как метод класса и т.д. В PHP нет перечислений (enum), нет событий (построение событий базируется на использовании внешних методов, например, через добавление функции в свойства тега, присоединение AddEventListener к объекту, но это приводит к проблемам из-за большого числа различных браузеров), нет возможности перегружать методы (вследствие отсутствия строгой типизации в языке) – это малая часть особенностей, которые возникают при использовании PHP относительно объектно-ориентированных языков. PHP имеет все преимущества функционального программирования к примеру функции высшего порядка, каррирование, простые функции. В PHP не реализована многопоточность, а также не поддерживаются или некорректно работают некоторые кодировки, к примеру, Unicode.

PHP является менее строгим языком, нежели .Net-языки, из-за отсутствия строгой типизации и отсутствия обязательного объявления переменных. В результате чего будут созданы различные опасные конструкции, которые не скомпилируются даже в компиляторе C, но при этом в PHP выполняться с непредвиденным результатом. В свою очередь в языках .Net программа всегда проверяется компилятором на ошибки.

Под язык PHP существует множество сред разработки: Zend Studio, Eclipse для PHP, плагин под Visual Studio и т. п. Все они являются довольно развитыми средами с поддержкой множества современных возможностей

сред разработки. Основной продукт, который используется для разработки веб-приложений на ASP.Net – это Microsoft Visual Studio. Есть и сторонние средства разработки к примеру, Visual Code, но они менее распространены.

По большому счету, все продукты как для PHP, так и для ASP.Net имеют встроенную подсказку (Intellisense), возможность доступа к базе данных, возможность отладки и т. п. Однако различия все же имееются. В разработке любых приложений очень помогает Intellisense при правильном ее использовании. Но в PHP она сама по себе не может так быть широко использована, как в ASP.Net – опять же, потому, что PHP – не типизированный язык.

В большинстве случаев совместно с PHP применяется СУБД MySql. А для ASP.Net-приложения используют сервер баз данных от Microsoft – Microsoft SQL Server. Обе технологии умеют работать с любыми СУБД независимо от производителя. Если сравнить MySQL и Microsoft SQL Server, то здесь с большим перевесом выигрывает сервер от Microsoft. И одна особенность MySQL (при использовании совместно с PHP) – наличие веб-интерфейса для доступа к базе данных (phpMyAdmin). В качестве визуального интерфейса для MSSql можно использовать SSMS, поставляемую вместе с ядром базы данных, либо воспользоваться сторонними инструментами к примеру, DbForge.

2.3 Анализ паттернов проектирования и выбор оптимального

Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования. Предполагается, что есть некоторый набор общих формализованных проблем, которые довольно часто встречаются, и паттерны предоставляют ряд принципов для их разрешения. При разработке программного обеспечения применение паттернов дает возможность формализовать проблему в виде классов, объектов и связей между ними. Затем применяется

один из существующих паттернов для ее решения и используется готовый шаблон.

Паттерны, как правило, не зависят от языка программирования. Их принципы применения будут аналогичны и в C#, и в Jave, и в других языках. Также паттерны упрощают коллективную разработку программ. Зная применяемый паттерн проектирования и его основные принципы другому программисту будет проще понять его реализацию и использовать ее. В основу классификации основных паттернов положена цель или задачи, которые определенный паттерн выполняет.

Порождающие паттерны абстрагируют процесс инстанцирования (порождения) классов и объектов. Среди них выделяют [9]:

- абстрактная фабрика (Abstract Factory);
- строитель (Builder);
- фабричный метод (Factory Method);
- прототип (Prototype);
- одиночка (Singleton).

Структурные паттерны рассматривают как классы и объекты образуют крупные структуры – более сложные по характеру классы и объекты. К таким шаблонам относятся [9]:

- адаптер (Adapter);
- мост (Bridge);
- компоновщик (Composite);
- декоратор (Decorator);
- фасад (Facade);
- приспособленец (Flyweight);
- заместитель (Proxy).

Поведенческие паттерны определяют алгоритмы и взаимодействие между классами и объектами [9]. Среди подобных шаблонов можно выделить:

– цепочка обязанностей (Chain of responsibility);
– команда (Command);
– интерпретатор (Interpreter);
– итератор (Iterator);
– посредник (Mediator);
– хранитель (Memento);
– наблюдатель (Observer).

Паттерны классов описывают отношения между классами посредством наследования. Отношения между классами определяются на стадии компиляции. К таким паттернам относятся [9]:

– фабричный метод (Factory Method);
– интерпретатор (Interpreter);
– шаблонный метод (Template Method);
– адаптер (Adapter);

Паттерны объектов описывают отношения между объектами, которые возникают на этапе выполнения и обладают большей гибкостью. К таким паттернам относятся следующие [9]:

– абстрактная фабрика (Abstract Factory);
– строитель (Builder);
– прототип (Prototype);
– одиночка (Singleton);
– мост (Bridge);
– компоновщик (Composite);
– декоратор (Decorator);
– фасад (Facade);
– приспособленец (Flyweight);
– заместитель (Proxy);
– цепочка обязанностей (Chain of responsibility);
– команда (Command).

Различных шаблонов проектирования гораздо больше приведенных выше паттернов. При выборе наилучшего шаблона надо выделить все используемые сущности и связи между ними и абстрагировать их от конкретной ситуации. Затем следует выявить, вписывается ли абстрактная форма решения задачи в некоторый паттерн. Например, при создании новых объектов наилучшими могут оказаться порождающие паттерны.

При использовании паттернов, как и при любом другом стиле программирования, следует придерживаться принципа KISS (Keep It Simple, Stupid) – сохранять код программы простым и ясным. Смысл паттернов состоит не в усложнении кода программы, а в его упрощении.

2.3.1 Паттерн «Наблюдатель» (Observer)

Назначение паттерна заключается в определении зависимости типа «один ко многим» между объектами так, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются. Наблюдатель уведомляет все заинтересованные стороны о произошедшем событии или об изменении своего состояния [9].

Существует два способа общения между двумя программными элементами. Компонент 1 может обратиться к Компоненту 2 для получения каких-то данных или выполнения некоторой операции. В этом случае Компонент 2 выполняет определенную работу, когда его об этом попросят. В некоторых случаях Компонент 2 является активным, содержит собственный поток исполнения или каким-то другим способом следит за своим состоянием. В этом случае Компонент 2 может уведомить Компонент 1 о некотором событии.

Первая модель взаимодействия называется pull-моделью, а вторая – push-моделью (рисунок 2.1).

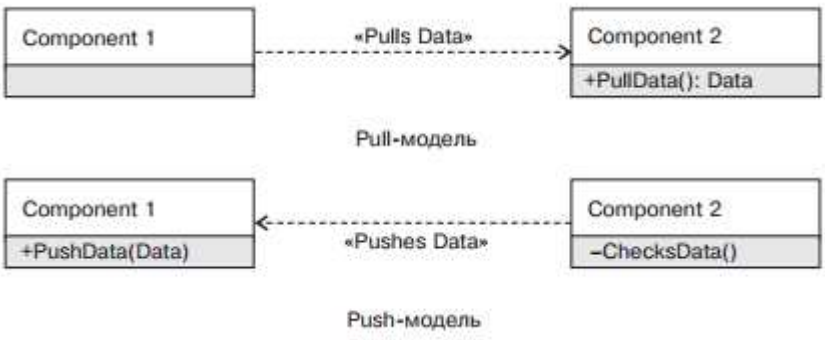

Рисунок 2.1 – Pull- и push-модель взаимодействия

Push-модель взаимодействия появилась задолго до распространения ООП и паттернов проектирования. В мире структурного программирования push-модель реализуется с помощью методов обратного вызова (callbacks). В функциональном программировании push-модель представлена в виде реактивной модели. При использовании объектно-ориентированного подхода push-модель реализуется с помощью паттерна «Наблюдатель».

2.3.2 Паттерн «Одиночка» (Singleton)

В автоматизированной системе могут существовать сущности в единственном экземпляре (система ведения системного журнала сообщений, драйвер дисплея и т. п.). В таких случаях необходимо уметь создавать единственный экземпляр некоторого типа, предоставлять к нему доступ извне и запрещать создание нескольких экземпляров того же типа. Для реализации таких целей служит паттерн Singleton.

Архитектура паттерна Singleton основана на идее использования глобальной переменной, имеющей свойства [9]:

1. Переменная доступна всегда (время жизни глобальной переменной – от запуска программы до ее завершения).

2. Переменная может быть доступна из любой части программы.

Однако, использовать глобальную переменную некоторого типа непосредственно невозможно, так как существует проблема обеспечения

единственности экземпляра. Для решения этой проблемы паттерн Singleton обеспечивает контроль над созданием единственного объекта через класс. Доступ к объекту осуществляется через статическую функцию – член класса, которая возвращает указатель или ссылку на него. Этот объект будет создан при первом обращении к методу, все последующие вызовы возвращают его адрес. Для обеспечения уникальности объекта, конструкторы и оператор присваивания объявляются закрытыми.

2.3.3Паттерн «Абстрактная фабрика» (Abstract Factory)

Существуют две классические разновидности паттерна: «Абстрактная фабрика» и «Фабричный метод». Обе разновидности предназначены для инкапсуляции создания объекта или семейства объектов. На практике очень часто отходят от классических реализаций паттернов и называют фабрикой любой класс, инкапсулирующий в себе создание объектов. Абстрактная фабрика предоставляет интерфейс для создания семейства взаимосвязанных или родственных объектов (dependent or related objects), не специфицируя их конкретных классов. Другими словами, абстрактная фабрика представляет собой стратегию создания семейства взаимосвязанных или родственных объектов [9].

Архитектурный подход, при котором реализовано прямое взаимодействие компонент-компонент считается плохой практикой и приводит к ошибкам. Концепция паттерна «наблюдатель» позволяет скоординировать изменения зависящих элементов. Предложенная концепция позволит избежать запутанности при росте компонентов пользовательского интерфейса.

2.4 Технология AJAX

Технология AJAX (Asynchronous JavaScript) представляет подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером [10].

Актуальные веб-приложения работают по технологии клиент-сервер (рисунок 2.2).

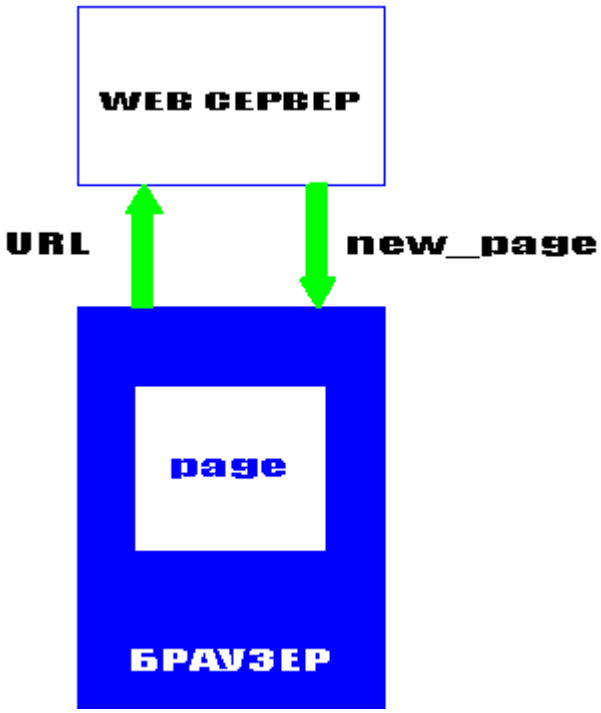

Рисунок 2.2 – Работа веб-программы по клиент-серверной технологии

Пользователь в веб-браузере открывает какую-либо веб-страницу page, на которой есть гиперссылки, ведущие на другие страницы. При нажатии на любую из них браузер посылает запрос URL на сервер, с которым связана эта ссылка. Если не существует сервера, связанного с этой ссылкой, или имеются проблемы связи с внутренней сетью, то браузер сгенерирует страницу, подобную показанной на рисунке 2.3.

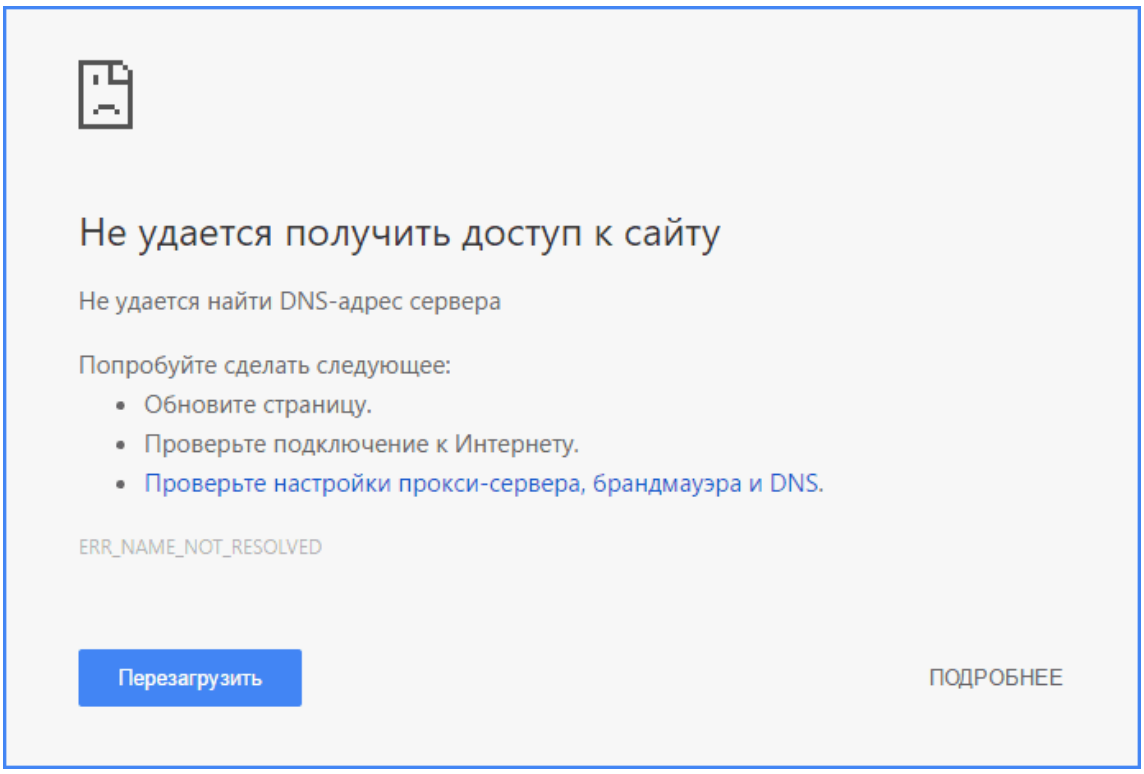

Рисунок 2.3 – Ошибка соединения

В случае существования сервера, но отсутствии на нем документа, указанного в запросе сервер сам, создаст HTML страницу с описанием ошибки. Если все выполнено верно, то в ответ сервер выдаст новую страницу. В любом случае, в браузер будет загружена новая страница *new_page*, что является существенным минусом данной технологии. Более того, работа ведется в синхронном режиме, т. е. после того как браузер отослал на сервер запрос он ожидает ответ, и пока он не получен ничего предпринимать не будет. Для обхода таких ожиданий можно использовать, например, DOM методы языка JavaScript, для динамического изменения фонового рисунка без перезагрузки страницы.

Очевидно, что клиент-серверная технология в классическом виде не способна удовлетворить все требования, которые предъявляются к современным веб-приложениям. Ключевым отличием AJAX технологии от клиент-серверной является наличие AJAX, состоящего из двух частей: клиент-приложение (на языке JavaScript) и сервер-приложение (на любом

серверном языке). Также реализуется другая логика взаимодействия приложения-клиента с сервером (рисунок 2.4).

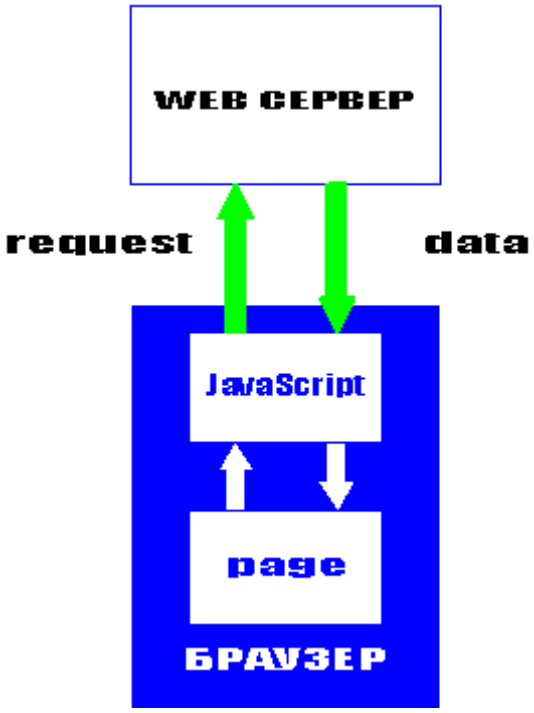

Рисунок 2.4 – Работа веб-программы по технологии AJAX

При активации какого-либо элемента управления интерфейса браузер не делает запрос новой страницы с сервера, а запускает клиентскую часть. Приложение-клиент обращается к серверу через запрос requst и запрашивает только те данные, которые должны измениться на странице. После получения данных data от приложения-сервера, клиент-приложение производит обновление части страницы через DOM методы без перегрузки всей страницы в целом. При этом возможна работа в асинхронном режиме.

Большой плюс данного подхода в том, что он не исключает работу по клиент-серверной схеме. То есть на одной и той же странице часть элементов управления может реализовывать клиент-серверную технологию, а часть – технологию AJAX.

При применении данной технологии в ходе разработке АСУ сетевого планирования позволило [11]:

1. Сэкономить трафик. Применение асинхронного метода позволило существенно снизить загружаемый трафик при работе с веб-приложением, по причине того, что при перезагрузке всей страницы, время отклика значительно увеличивается, а с применением AJAX загрузится только обновленная часть веб-страницы.

2. Уменьшить нагрузку на сервер. При правильной реализации, AJAX позволяет снизить нагрузку на сервер в несколько раз. В частности, все страницы сайта чаще всего генерируются по одному шаблону, включая неизменные элементы («шапка», «навигационная панель», «подвал» и т. д.), для генерации которых требуются обращения к разным файлам, время на обработку скриптов (а иногда и запросы к БД) – всё это можно опустить, если заменить полную загрузку страницы генерацией и передачей лишь содержательной части. Дизайн страницы также обычно содержит множество файлов, связанных с оформлением (картинки, стили), на повторную обработку которых не надо тратить время, используя AJAX (экономия на количестве HTTP-соединений значительно выгоднее, чем на сокращении трафика каждого из них).

3. Ускорить реакцию интерфейса. Поскольку загрузка изменившейся части значительно быстрее, то пользователь видит результат своих действий быстрее и без мерцания страницы (возникающего при полной перезагрузке).

4. Осуществлять интерактивную обработку данных. Например, при вводе поискового запроса в какую-либо систему поиска выводится подсказка с возможными вариантами запроса. На многих сайтах при регистрации пользователь вводит имя, и сразу же видит, доступно это имя или нет. AJAX удобен для программирования административных панелей и других инструментов, которые выводят меняющиеся со временем данные.

2.5 API – Application programming interface

Интерфейс программирования приложений (интерфейс прикладного программирования, API) представляет собой набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. API определяет функциональность, которую предоставляет программа (модуль, библиотека), при этом позволяет абстрагироваться от того, как именно эта функциональность реализована.

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию – высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов. API библиотеки функций и классов включает в себя описание сигнатур и семантики функций.

Сигнатура функции – часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других. В различных языках программирования существуют разные представления о сигнатуре функции, что также тесно связано с возможностями перегрузки функций в этих языках. Различают сигнатуру вызова и сигнатуру реализации функции. Сигнатура вызова обычно составляется по синтаксической конструкции вызова функции с учётом сигнатуры области видимости данной функции, имени функции, последовательности фактических типов аргументов в вызове и типе результата. В сигнатуре реализации обычно участвуют некоторые элементы из синтаксической конструкции объявления функции: спецификатор области видимости функции, её имя и последовательность формальных типов аргументов [12].

Например, в языке программирования C++ простая функция однозначно опознаётся компилятором по её имени и последовательности типов её аргументов, что составляет сигнатуру функции в этом языке. Если

функция является методом некоторого класса, то в сигнатуре будет участвовать и имя класса.

В языке программирования Java сигнатуру метода составляет его имя и последовательность типов параметров; тип значения в сигнатуре не участвует.

Семантика функции – это описание того, что данная функция делает. Семантика функции включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Обычно результат выполнения зависит только от значений аргументов функции, но в некоторых модулях есть понятие состояния. Тогда результат функции может зависеть от этого состояния, и, кроме того, результатом может стать изменение состояния. Логика этих зависимостей и изменений относится к семантике функции. Полным описанием семантики функций является исполняемый код функции или математическое определение функции [12].

Применение WEB API при реализации автоматизированной системы (АС) сетевого планирования и управления (СПУ) позволило:

- исключить дублирование исходного кода;
- получить единый интерфейс по получению данных;
- построить функциональность, которая может быть использована в других приложениях и иных целях.

2.6 Выводы по главе 2

На сегодняшний день у разработчиков веб-приложений есть богатый выбор относительно того, какой язык (или технологию) использовать для создания приложения. Вариантов много: Perl, PHP, ASP, ASP.Net, JSP. Самыми распространенными (с большим отрывом от остальных) сегодня являются PHP и ASP.Net.

В результате изучения предметной области и различных технологий реализации информационных систем с Web-доступом было решено:

1) выбрать в качестве технологии реализации платформу ASP.NET, в связи тем, что ASP.NET ориентирована на большие проекты, а PHP для более мелких проектов. Компиляция программ. В отличие от PHP, в .NET код компилируется, благодаря чему он исполняется гораздо быстрее. Так же ASP.NET чуть медленнее работает с малой нагрузкой, с большой за счет компиляции работает быстрее. У PHP – множество шаблонизаторов на выбор, у ASP.NET шаблонизация заложена в основу;

2) Наиболее подходящим шаблоном проектированием является Observer (Наблюдатель) в связи необходимостью постоянного контроля над длительностями на диаграмме Ганта.

3 Разработка приложения

3.1 Выявление сущностей и построение диаграммы классов предметной области

Для отображения статической структуры модели системы необходимо построить диаграмму классов. Диаграмма классов языка UML представляет собой ориентированный граф, на котором представлена совокупность декларативных или статических элементов модели, таких как классы с атрибутами и операциями, а также связывающие их отношения. Диаграмма классов может содержать интерфейсы, пакеты, отношения, отдельные экземпляры классификаторов, такие как объекты и связи.

Для реализации проекта автоматизированной системы, была получена объектная модель, представленная на рисунке 3.1, которую после заполнения необходимо развернуть в программный код. Объектно-ориентированная технология основывается на объектной модели, основными принципами которой являются абстракция, инкапсуляция, модульность, иерархичность, типизация, параллелизм и сохраняемость. Каждый из этих принципов сам по себе не нов, но в объектной модели они впервые применены в совокупности.

Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя. Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности поведения от несущественных. Инкапсуляция – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации. Модульность – это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули. Таким образом, принципы абстрагирования, инкапсуляции и модульности являются взаимодополняющими. Объект

логически определяет границы определенной абстракции, а инкапсуляция и модульность делают их физически неизменяемыми.

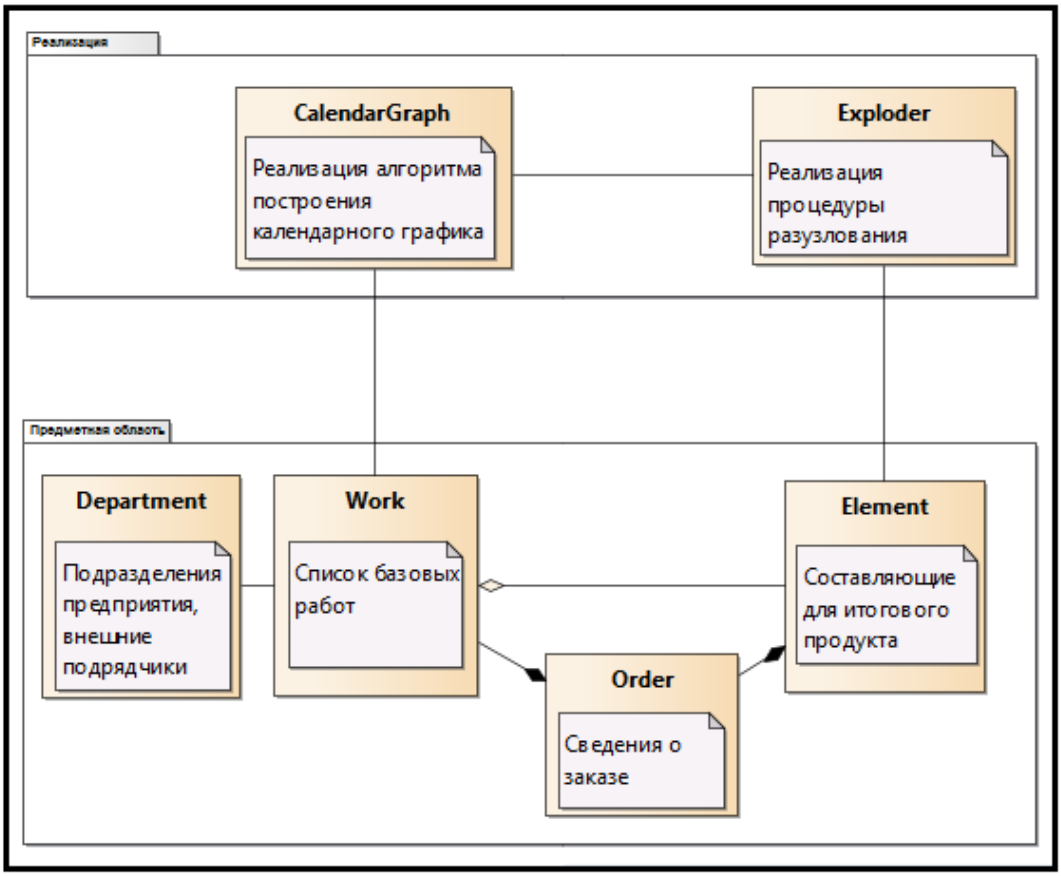

Рисунок 3.1 – Объектная модель

Значительное упрощение в понимании сложных задач достигается за счет образования из абстракций иерархической структуры. Таким образом иерархия – это упорядочение абстракций, расположение их по уровням. Типизация является способом защиты от использования объектов одного класса вместо другого, или по крайней мере, управлять таким использованием. Типизация заставляет выражать абстракции так, чтобы язык программирования, используемый в реализации, поддерживал соблюдение принятых проектных решений. В то время, как объектно-ориентированное программирование основано на абстракции, инкапсуляции и наследовании, параллелизм главное внимание уделяет абстрагированию и синхронизации

процессов. Каждый объект (полученный на конкретном уровне абстракции) может представлять собой отдельный поток управления (абстракцию процесса). Такой объект называется активным. Система, построенная на основе ООП, может быть представлена, как совокупность взаимодействующих объектов, часть из которых является активной и выступает в роли независимых вычислительных центров [13].

Использование объектной модели приводит к построению систем на основе стабильных промежуточных описаний, что упрощает процесс внесения изменений. Это дает системе возможность развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

Объектная модель уменьшает риск разработки сложных систем, прежде всего потому, что процесс интеграции растягивается на все время разработки, а не превращается в единовременное событие. Объектный подход состоит из ряда хорошо продуманных этапов проектирования, что также уменьшает степень риска и повышает уверенность в правильности принимаемых решений.

На основе анализа предметной области и объектной модели предоставленной аналитиком, выделим следующие классы необходимые для работы нашей системы:

1. Класс Element (рисунок 3.2). Исходное понятие производственного процесса, представляющее собой единицу продукции (составную или элементарную определенного типа), на которую составляется конструкторская документация и техпроцесс. Исследование предметной области выявило, что все изготавливаемые позиции имеют наименование по схеме ИНД + ОБОЗН и относятся к одному из следующих типов:

– Документация. Единица конструкторской, технологической, справочной или эксплуатационной документации для изделия;

– Блок. Составная единица, включающая в свой состав несколько более примитивных единиц;

– Деталь. Элементарная позиция, не включающая в свой состав иных элементов;

– Нормализованная позиция. Элемент, чье обозначение в силу особенностей хранения данных в отделе АСУП, было нормализовано к определенной форме;

– Покупные комплектующие изделия. Простые элементы, изготавливаемые за пределами предприятия.

На основе вышеизложенного, определяем основные моменты, характеризующие класс, который будет представлять сущность элемента:

– Составное формирование наименования из текстовых полей индекса и обозначения;

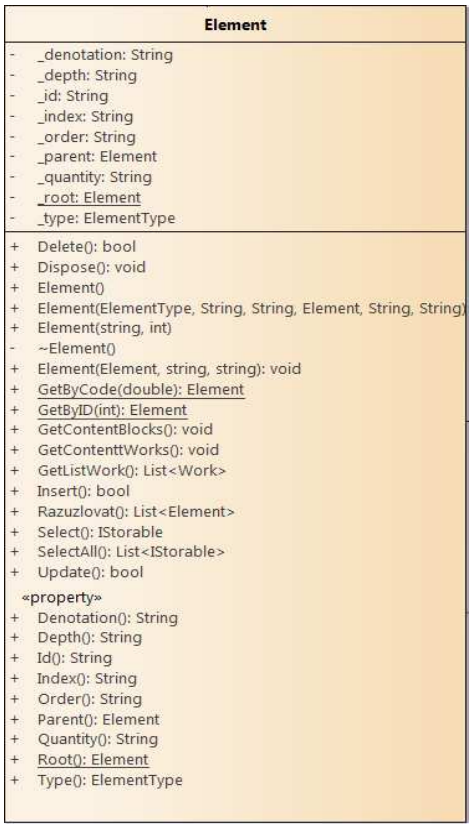

Рисунок 3.2 – Класс «Элемент»

– Присутствие в каждом объекте списка объектов того же типа, представляющих состав изделия;

– Присутствие в каждом объекте списка объектов типа «Работа», показывающий набор операций для изготовления данного блока;

– Реализация поведения: добавления, удаления, изменения позиции, а также вызов метода разузлования для получения состава блока.

2. Класс Order (Заказ). По номеру заказа задается начальная точка построения календарного графика.

3. Заказ обладает функционалом: обновление, удаление, а также возможность расчета длительности выполнения и окончания.

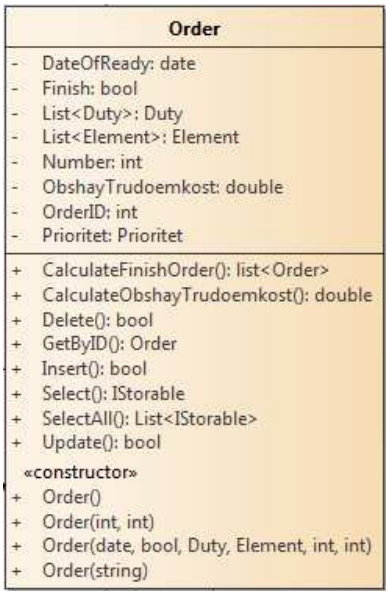

Рисунок 3.3 – Класс «Заказ»

4. Класс Work (Работа). Составной процесс, для которого необходимо, чтобы был сформирован класс «элемент», содержащий сведения о составе и устройстве изделия, а так же подсчитаны трудозатраты на выполнение изделия.

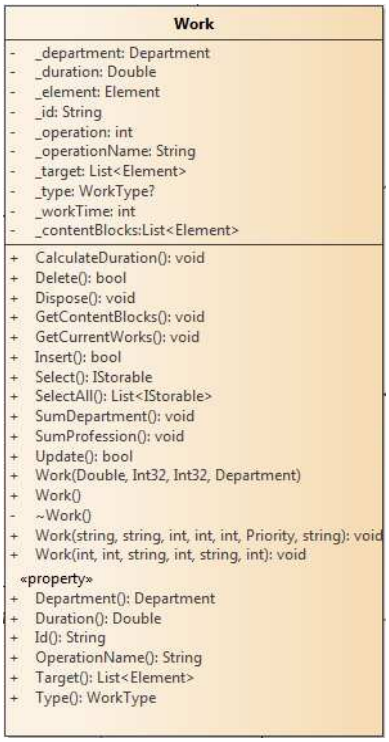

Рисунок 3.4 – Класс «Работа»

5. Класс Department (Цех). Класс, отображает подразделение, которое занимается изготовлением элемента. Обладает цветовым входом, отображаемым на графике.

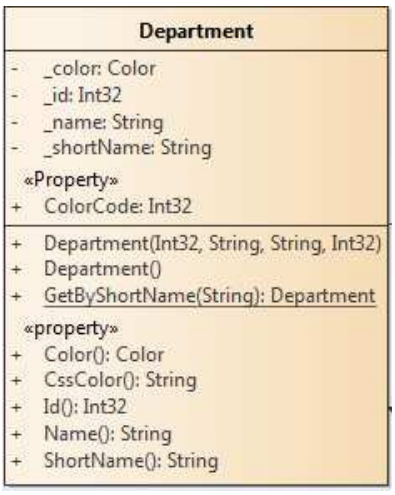

Рисунок 3.5 – Класс «Цех»

6. Класс Exploder (Разузлование). Этот класс отображает всю структуру заказа, начиная от головных блоков и заканчивая последним уровнем

вложенности. Принимая в конструкторе номер заказа, он рекурсивно выстраивает линеаризованное дерево с полным списком позиций.

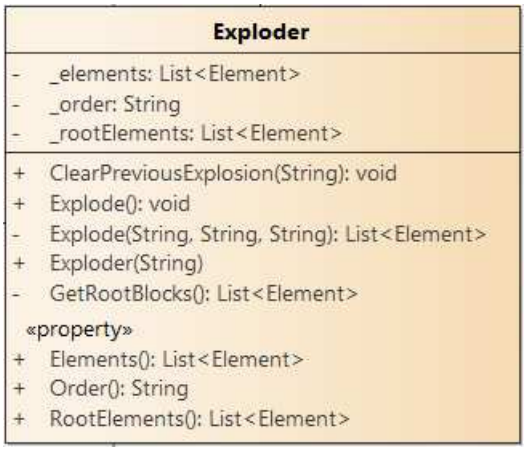

Рисунок 3.6 – Класс «Разузлование»

7. Класс CalendarGraph (Календарный график). Этот класс включает в себя сведения о заказе, элементах, всех работ и определяет дату начала

Рисунок 3.7 – Класс «Календарный график»

выполнения заказа. Используется как отправная точка для процесса построения календарного графика. Он содержит в себе дерево состава заказа и дерево работ.

3.2 Выявление основных расчетных операций

Для построения календарного графика и получения списка работ необходимо получить список блоков, используемых при сборке изделия. Для этого необходимо прибегнуть к рекурсивной функции разузлования, которая построит для заказа иерархическое дерево всех входящих позиций.

Разузлование – иерархическое разбиение изделия на входящие в него узлы, сборки, детали и расходные материалы [13].

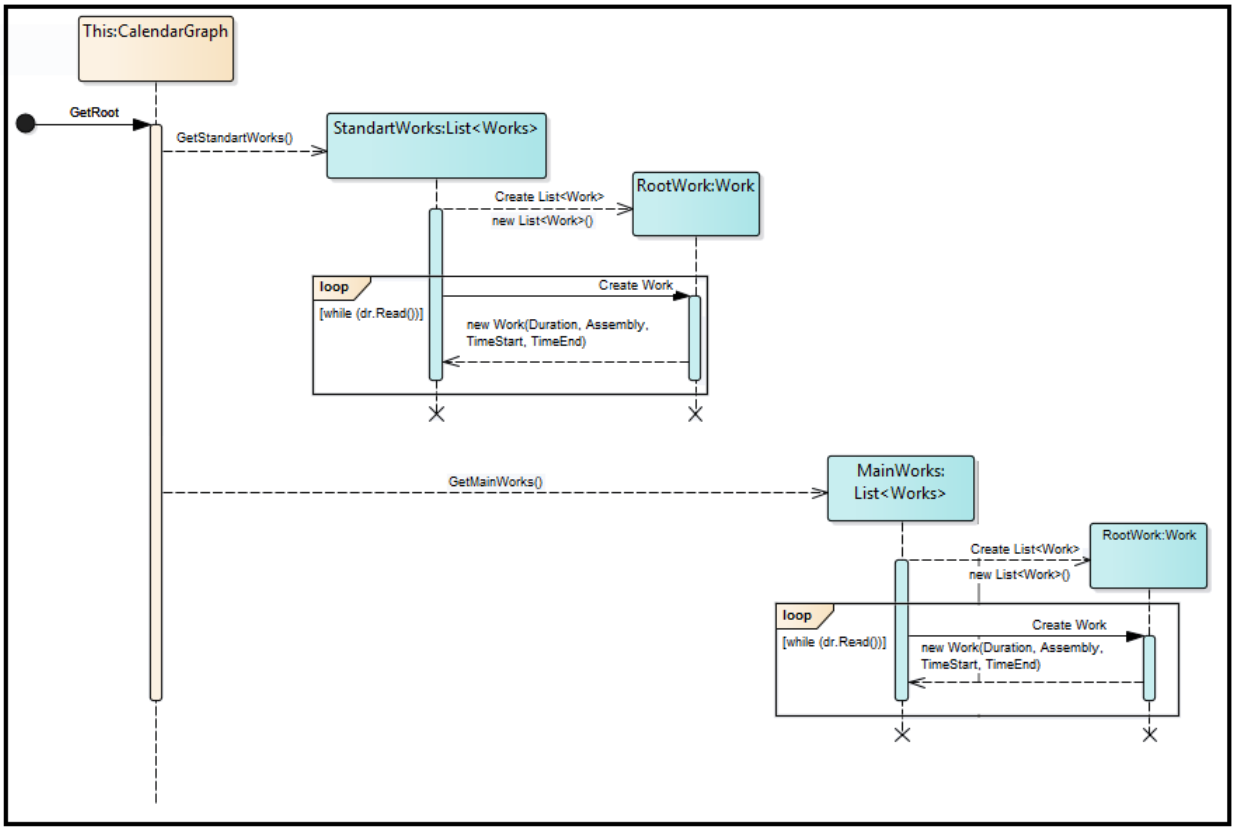

Рисунок 3.8 – Диаграмма последовательности, описывающая процесс построения графика агрегированных работ

Чтение разузлования заказа начинается с головных блоков и их заполнением и разузлованием головных блоков на позиции, которые также заполняются.

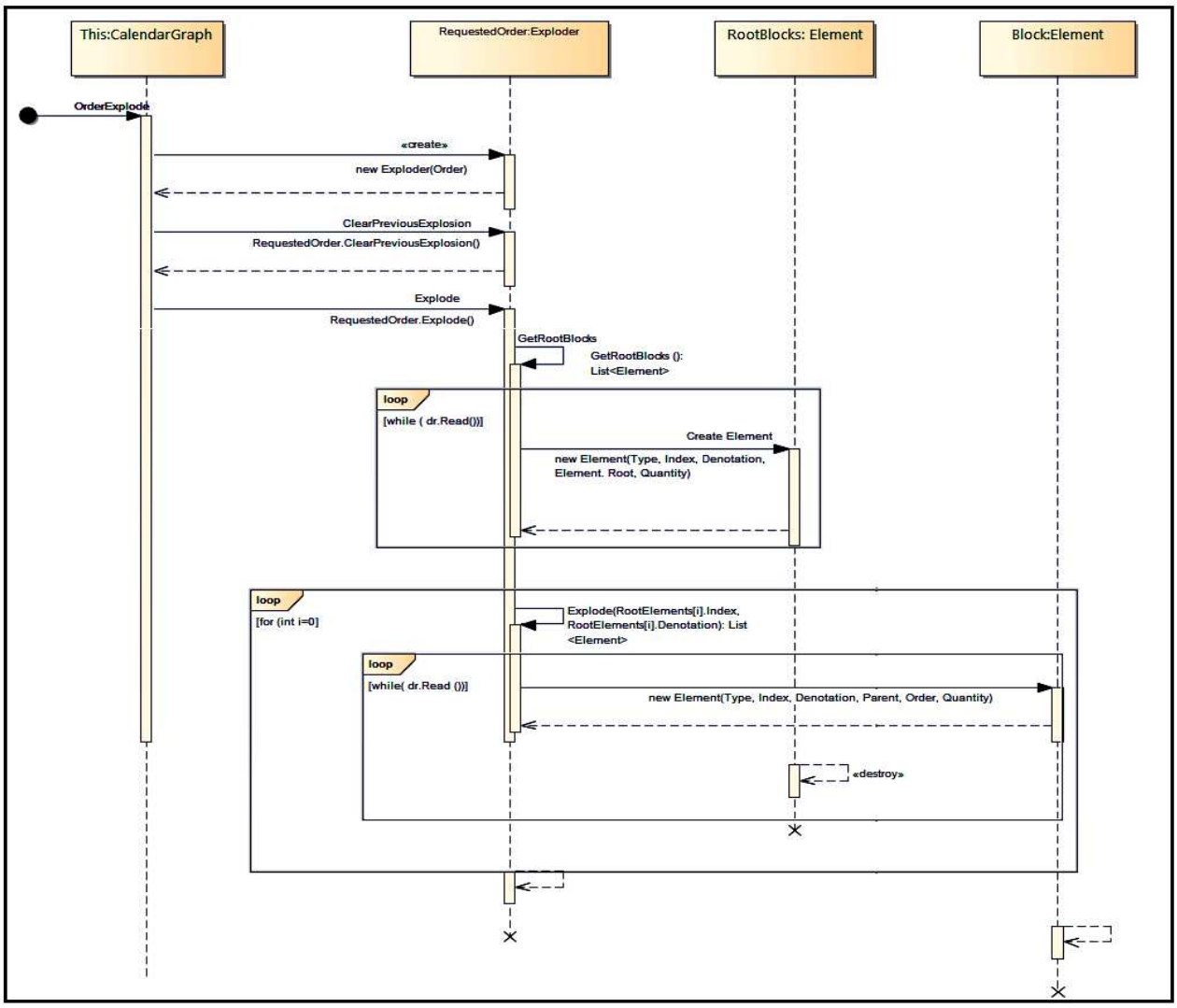

Рисунок 3.9 – Диаграмма последовательности, описывающая метод разузлования заказа

Диаграмма на рисунке 3.8 описывает работы, включенные в календарный график: стандартные, основные и представляет собой алгоритм перехода между этими типами работ.

На рисунке 3.9 изображена диаграмма последовательности, выполненная методом разузлования (OrderExplode) и представляющая собой алгоритм перехода между стандартными и основными типами работ. Результатом работы является полный список разузлования.

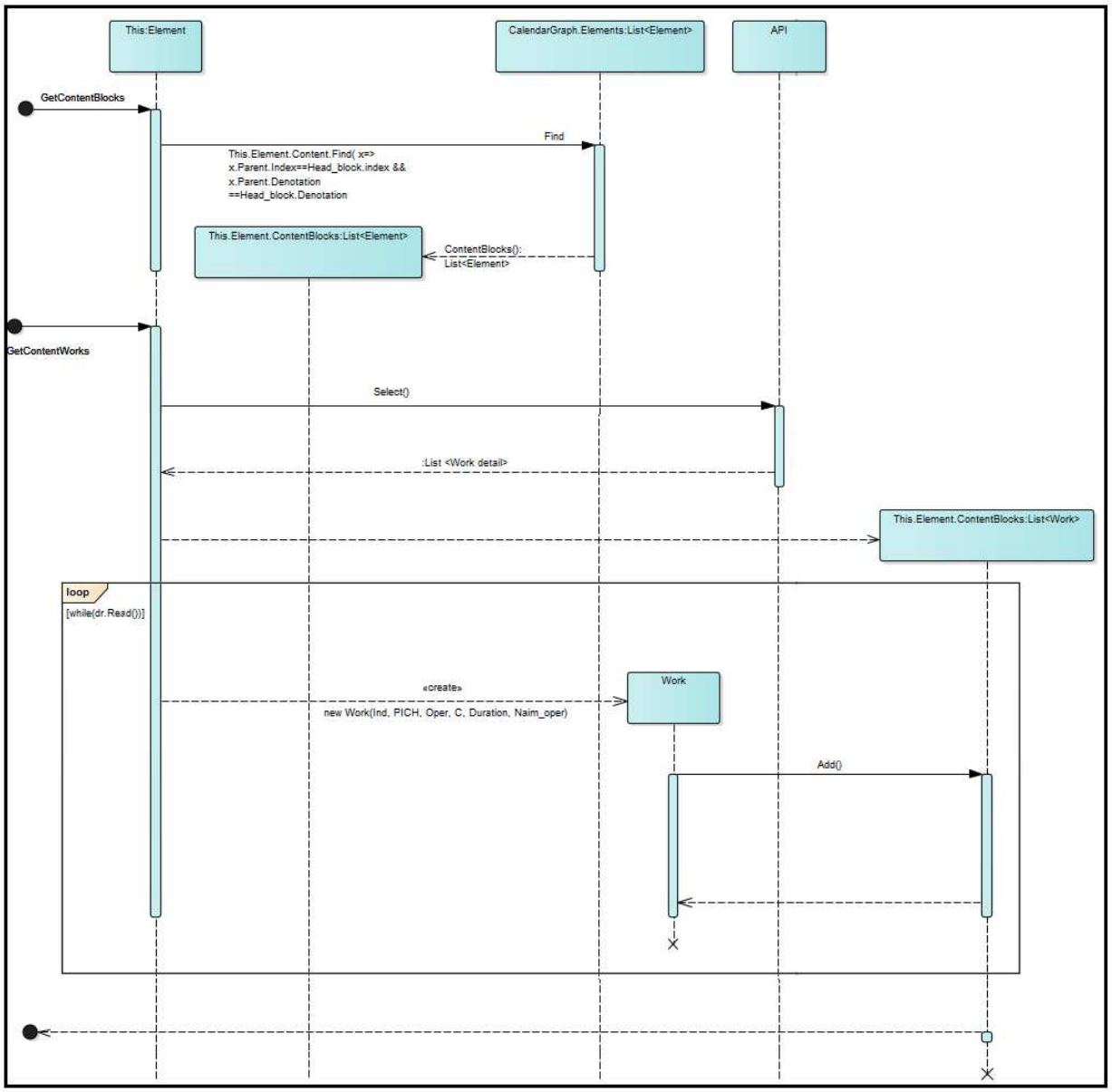

Рисунок 3.10 – Диаграмма последовательности, описывающая процесс развертывания агрегированной работы

Результатом выполнения диаграммы изображенной на рисунке 3.10 является получение списка работ и соответствующих длительностей изделий, объединенных по признаку – цех.

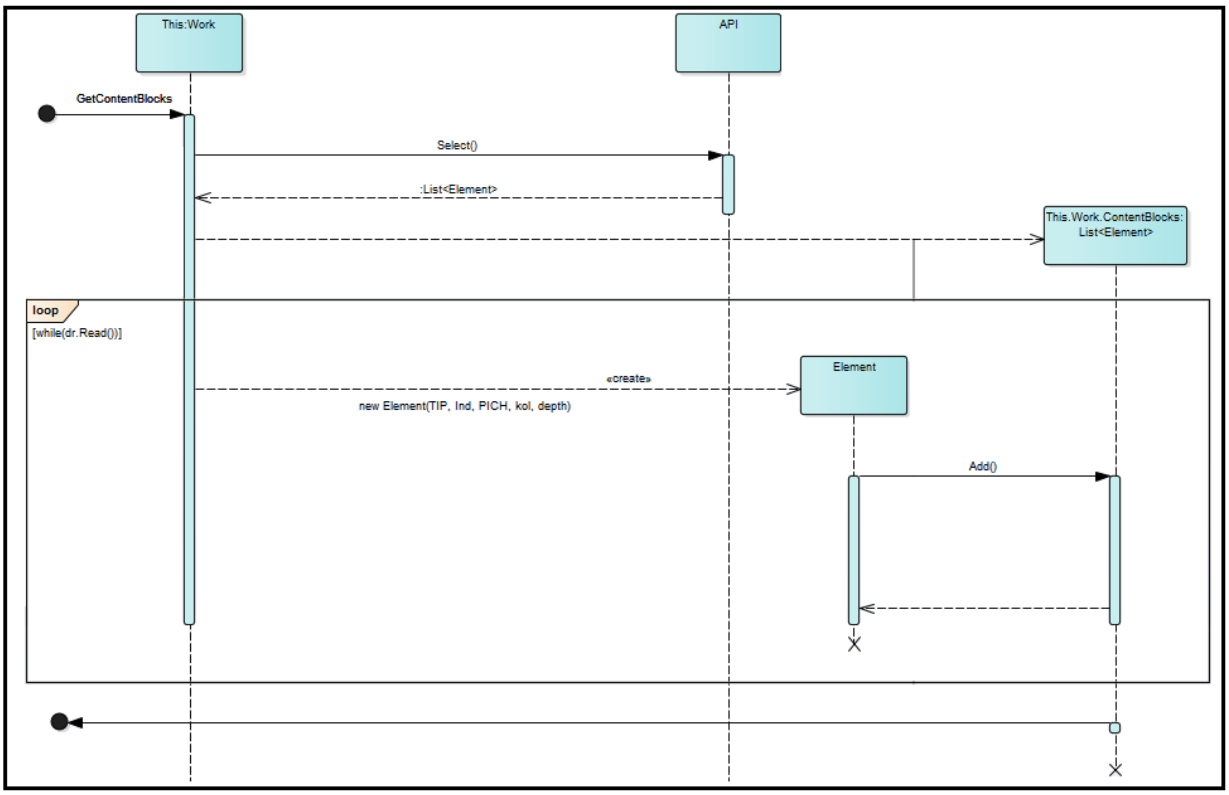

Рисунок 3.11 – Диаграмма последовательности, описывающая процесс получения состава блока на основе ее соотнесения с операцией

Диаграмма на рисунке 3.11 описывает событие после щелчка на головной блок в обозревателе заказа.

Разработанные диаграммы предназначены как для визуализации и эмпирической оценки поведенческих особенностей разработанного приложения, так и для численной оценки различных характеристик выбранного метода сетевого планирования. В данном случае такие оценки не приводятся ввиду их избыточности для решения задачи создания приложения. В исследовательских целях численное моделирование поведения приложения было проведено, показало конечность и однозначность выполнения разработанных алгоритмов. Стоит отметить, что, несмотря на наличие ряда циклических операций, присутствующих на некоторых диаграммах последовательности, укрупненная процедура построения сетевой модели не содержит значительного количества циклических операций. Таким образом, выбранный метод построения

сетевой модели вполне пригоден для реализации и внедрения в процесс производственного планирования.

3.3 Разработка MVC-приложения

Опираясь на приведенные в разделе 3.2. диаграммы, проведем разработку MVC-приложения сетевого планирования. Model-view-controller (MVC, «модель-представление-контроллер», рисунок 3.12) – схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Такая схема проектирования часто используется для построения архитектурного каркаса программного продукта [14].

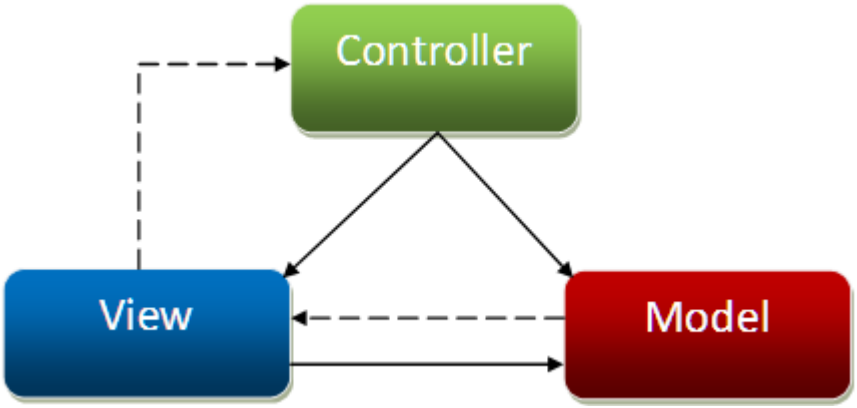

Рисунок 3.12 – схема MVC

Идея, которая лежит в основе конструкционного шаблона MVC заключается в чётком разделении ответственности за различное функционирование в приложениях (рисунок 3.13).

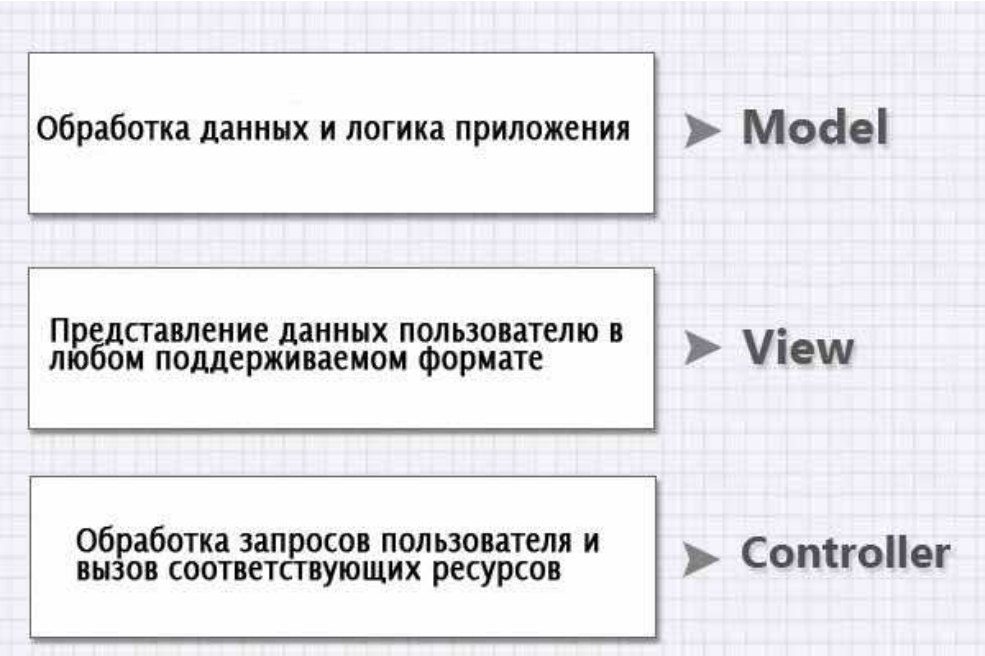

Рисунок 3.13 – Трехуровневая система

Контроллер (Controller) управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция – вызывать и координировать действие необходимых ресурсов и объектов, необходимых для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид [15].

Модель (Model) – правила, использующиеся для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определенным образом. Модель дает контроллеру представление данных, которые запросил пользователь. Модель данных будет одинаковой, вне зависимости от способа представления их пользователю. Модель содержит наиболее важную часть логики разрабатываемого приложения. Контроллер содержит в основном организационную логику для самого приложения [15].

Вид (View) обеспечивает различные способы представления данных, которые получены из модели. Вид может быть шаблоном, который заполняется данными, а может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб-приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

3.3.1 Уровень Model

Уровень модели, в целом, повторяет собой структуру, полученную при разработке диаграммы классов. В нем можно выделить четыре подуровня:

1. Служебный, или утилитарный

– DBManager.cs – файл, реализующий логику взаимодействия с базой данных, состояние подключение и т. д.;

– UserContext.cs – файл, реализующий Object-Relational Mapping (ORM) в приложении;

– Utils.cs – файл, содержащий чисто служебные методы и методы расширения (расширение функционала работы с перечислениями, классом цветов и т. д.).

2. Прикладной

– CalendarGraph.cs – файл, реализующий основные функции календарного графика (хранит в себе список корневых работ, результаты разузлования, ссылку на заказ и т. д.);

– Exploder.cs – файл, реализующий разузлование блоков.

3. Уровень данных

Содержит классы, спроецированный с помощью ORM из базы данных – элементы, работы, различного рода словари и т. д.

4. Уровень моделей представления

В проекте используются строго типизированные представления, поэтому каждое представление и частичное представление работает с моделью своего типа, реализованного на этом уровне. Список моделей предоставления повторяет список представлений.

3.3.2 Уровень View. Применение шаблона проектирования и Ajax технологии

Уровень представления отвечает за визуальную составляющую приложения и обработку логики взаимодействия пользователя с элементами графического интерфейса. Обобщенно его структура представлена в виде диаграммы компонентов, приведенной на рисунке 3.14.

В терминах фреймворка Microsoft MVC, уровень представления состоит из базовых элементов, приведенных в таблице 3.1.

Таблица 3.1 – Описание уровня представления

Название	Расширение	Описание
Представление	.cshtml	Файлы с HTML-разметкой, дополненные Razor-семантикой. Последняя добавляет динамическую составляющую на страницу, позволяя привязываться к модели/модели представления.
Частичное представление	.cshtml	Содержат только разметку данных, встраиваются в файлы представления. Поддерживают Razor-семантику.
Таблицы стилей	.css	Файлы стилей, выполняют две функции – содержат каскадные таблицы стилей, описывающие, как браузер должен отображать элементы на странице, а также предоставляет селекторы для JQuery-запросов
Скрипт	.js	Файлы с кодом на языке JavaScript, реализующий логику уровня клиента – в них содержатся обработчики пользовательского функционала (мышь, клавиатуры, валидаторы и т. д.), а также AJAX-запросы к контроллерам.

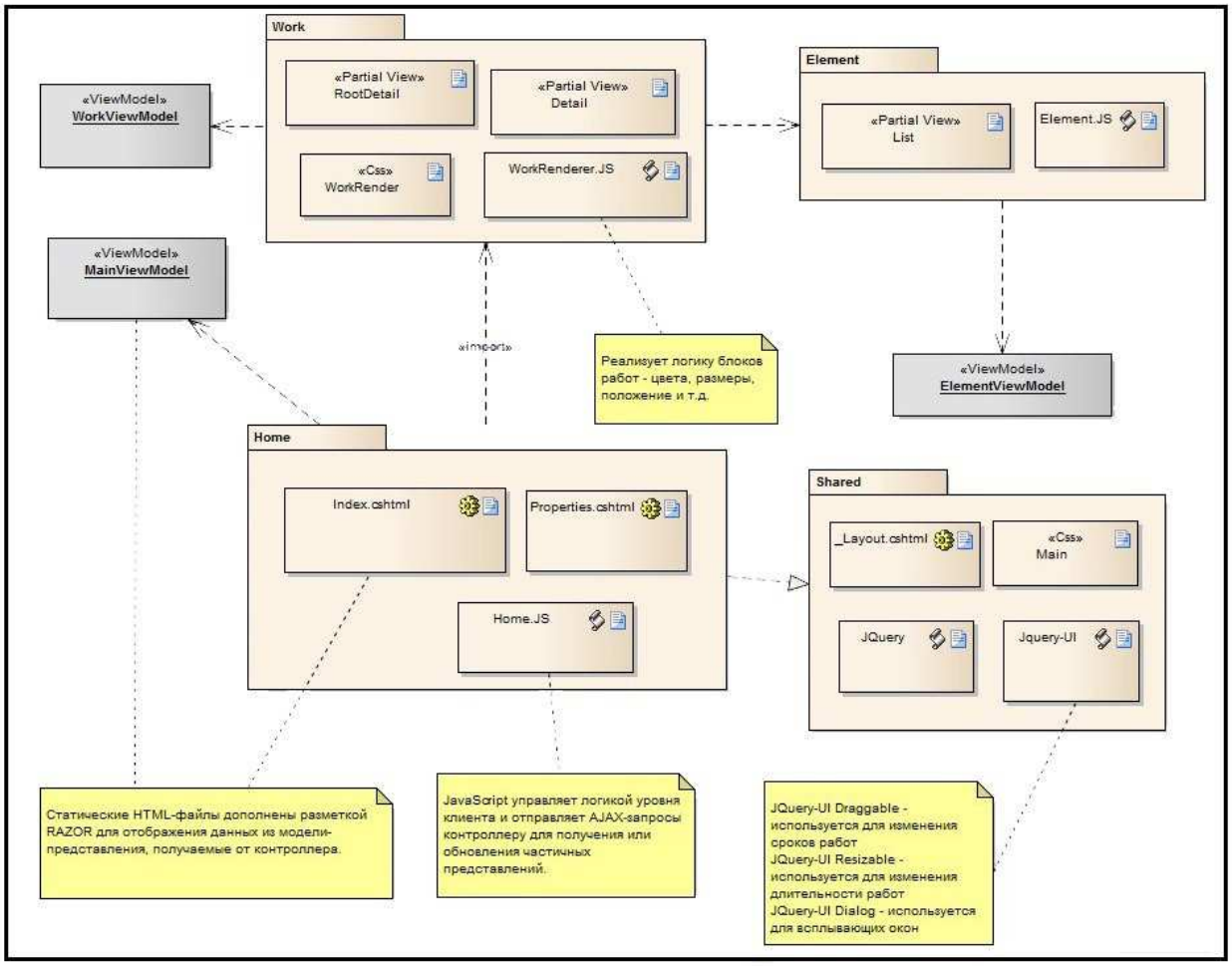

Рисунок 3.14 – Структура уровня представления

Основное представление, названное «домашней страницей» (Index) содержит основную HTML – разметку – для главной рабочей области и страницы с настройками. Кроме того, оно содержит (в библиотеке Init.JS) все обработчики, доступные с главного представление.

Главная рабочая область представлена из базового HTML – каркаса и двух @foreach-блоков, привязанных к соответствующим спискам модели представления – StandartWorks, MainWorks для стандартных и всех остальных работ соответственно. Привязка осуществляется следующим образом:

1. Загружается HTML страница.

2. Находится элемент с идентификатором GantHere var ContainerId = "gantt_here".

3. Происходит инициализация диаграммы gantt.init(ContainerId) (рисунок 3.15).

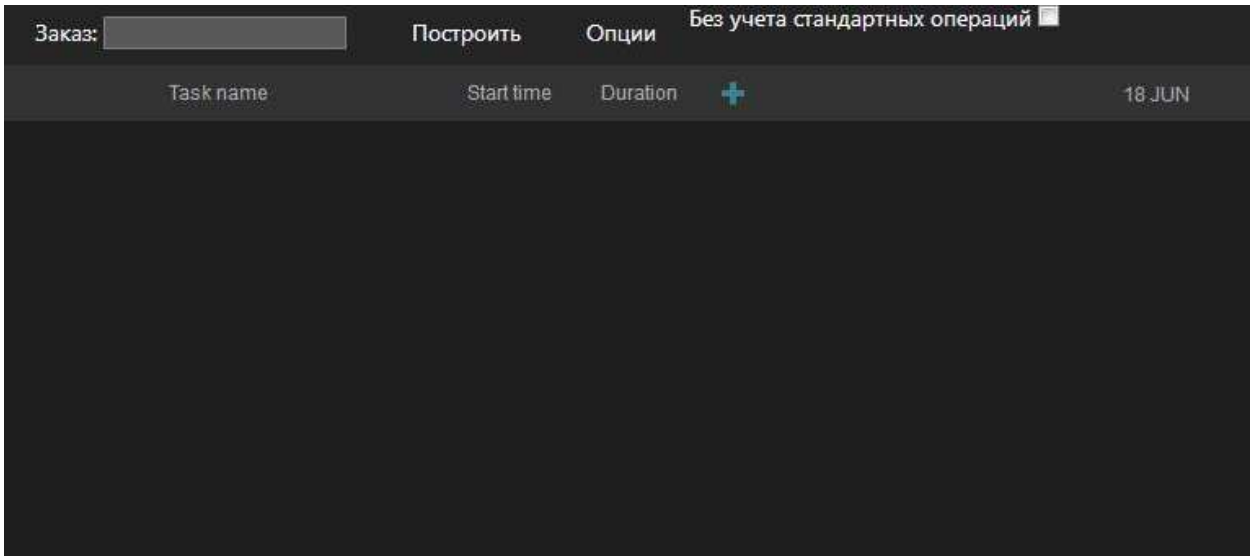

Рисунок 3.15 – Инициализация диаграммы

4. Задаются свойства диаграммы для локализации:

```
gantt.locale = {
        date: {
            month_full: ["Январь", "Февраль", "Март", "Апрель", "Мая", "Июнь", "Июль",
            "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"],
            month_short: ["Янв", "Фев", "Март", "Апр", "Май", "Июнь", "Июль", "Авг", "Сент",
            "Окт", "Нояб", "Дек"],
            day_full: ["Воскресенье", "Понедельник", "Вторник", "Среда", "Четверг", "Пятница",
            "Суббота"],
            day_short: [ "Вс", "Пн", "Вт", "Ср", "Чт", "Пт", "Сб"]
        },
        labels: {
            new_task: "New",
```

```
        icon_save: "Save",
        icon_cancel: "Cansel",
        icon_details: "Детали",
        icon_edit: "Изменить",
        icon_delete: "Удалить",
        confirm_closing: "",//Your changes will be lost, are you sure ?
        confirm_deleting: "Task will be deleted permanently, are you sure?",

        section_description: "Описание",
        section_time: "Период",
        column_duration: "Длительность",
        column_start_date: "Нач. дата",
        column_text: "Сборка",
        / link confirmation /
        confirm_link_deleting: "Dependency will be deleted permanently, are you sure?",
        link_from: "From",
        link_to: "To",
        link_start: "Start",
        link_end: "End",
        minutes: "Минуты",
        hours: "Часы",
        days: "Дни",
        weeks: "Неделя",
        months: "Месяц",
        years: "Год"
    }
};
```

5. Далее в поле ввода заказа вводится номер заказа и по клику «построить» (рисунок 3.16) выполняется метод Build, который располагается в атрибутах тега кнопки:

```
<div class="NavigatorInput">
    Заказ: <input id="RequestedOrderInput" name="Order" type="text" />
<div class="HLButtonStatic" onclick="Build()">Построить</div>
function Build()
{
  RequestedOrder = document.getElementById("RequestedOrderInput").value;
}
```

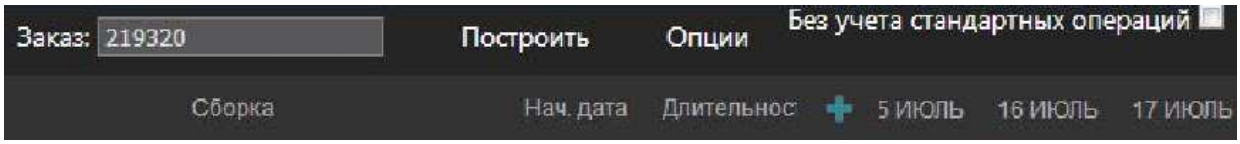

Рисунок 3.16 – Метод Build

6. После нажатия на кнопку находится значение элемента с идентификатором RequestedOrderInput и происходит выполнение трёх асинхронных запросов. Первый, заполняет данными диаграмму Ганта (рисунок 3.17):

– AjaxRequest("api/order/" + RequestedOrder + "/build", null, ShowDetails);

```
function ShowDetails(result)
{
  console.log(typeof result);
  var data = JSON.stringify({ data: result });
  gantt.parse(data);
}
```

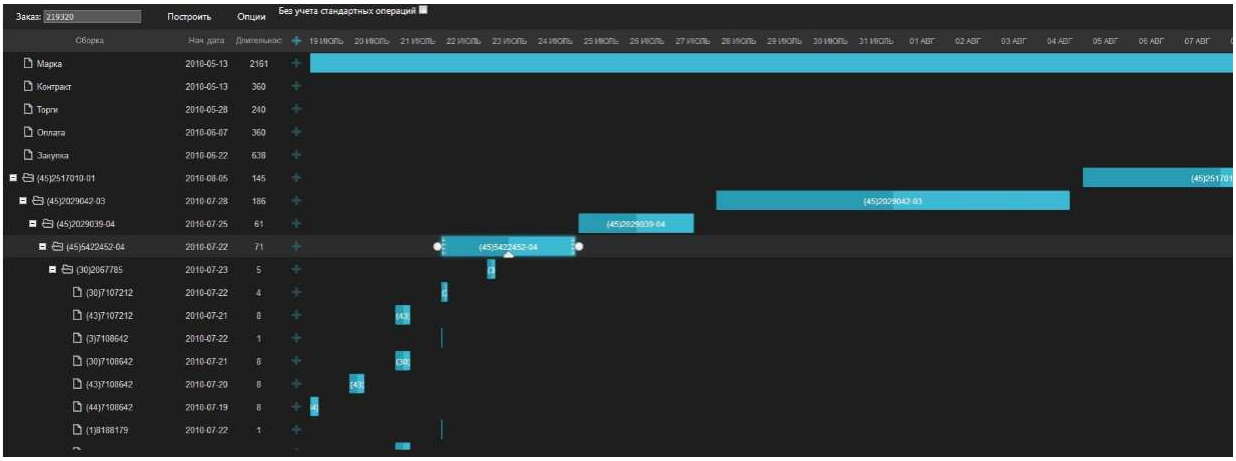

Рисунок 3.17 – Заполнение данными диаграммы Ганта

Щелчок по блоку вызывает обработчик, который создает асинхронный запрос к контроллеру, параметром передается идентификатор Id, полученный JQuery – селектором. Методы построение состава заказ по длительностям приведены в приложении А. Далее функция внутри контроллера возвращает частичное представление. Дальнейшие операции выполняются аналогично.

Второй запрос отображает на диаграмме состав изделия по уровням (рисунок 3.18):

– AjaxRequest("api/order/" + RequestedOrder + "/-1" + "/Exploder", null, Exploder);

```
function Exploder(result)
{
   tE.clear().draw();
   if (result.length !== 0)
{
      for (var i = 0; i < result.length; i++)
{
         tE.row.add([result[i].id_record, result[i].Type, result[i].Ind, result[i].Denotation, result[i].Amount, result[i].Depth]).draw();
      }
```

}

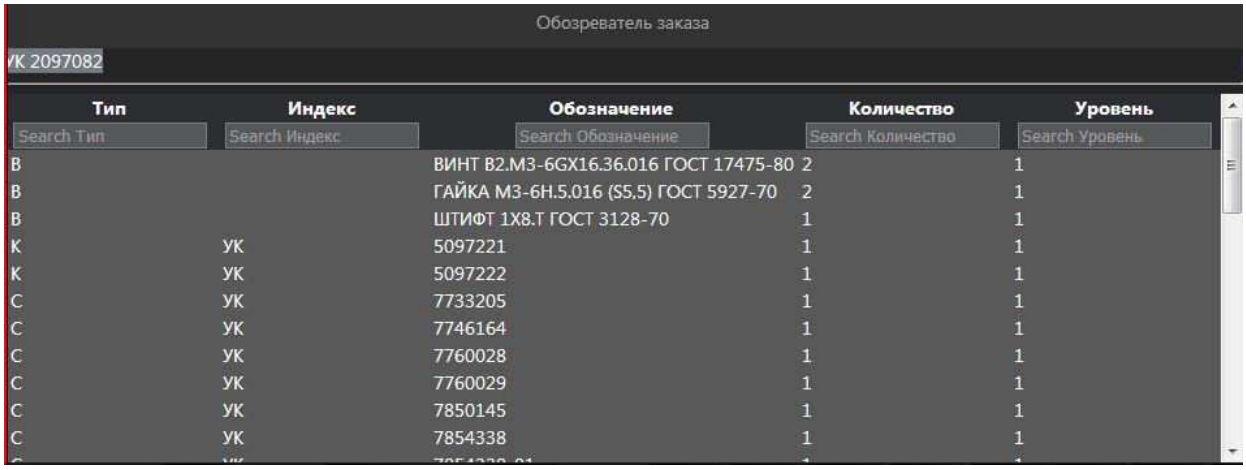

Рисунок 3.18 – Обозреватель заказа

Третий – детализацию работ (рисунок 3.19):

– AjaxRequest("api/Order/" + idRecord + "/WorkDetail", null, WorkDetail);

```
function WorkDetail(result)
{
   tW.fnClearTable();
   if (result.length !== 0)
      tW.fnAddData(result);
}
```

Рисунок 3.19 – Детализация работ

На рисунке 3.20 изображен дополнительный функционал календарного графика, а именно возможность ручного редактирования стандартных работ, которые устанавливаются по умолчанию.

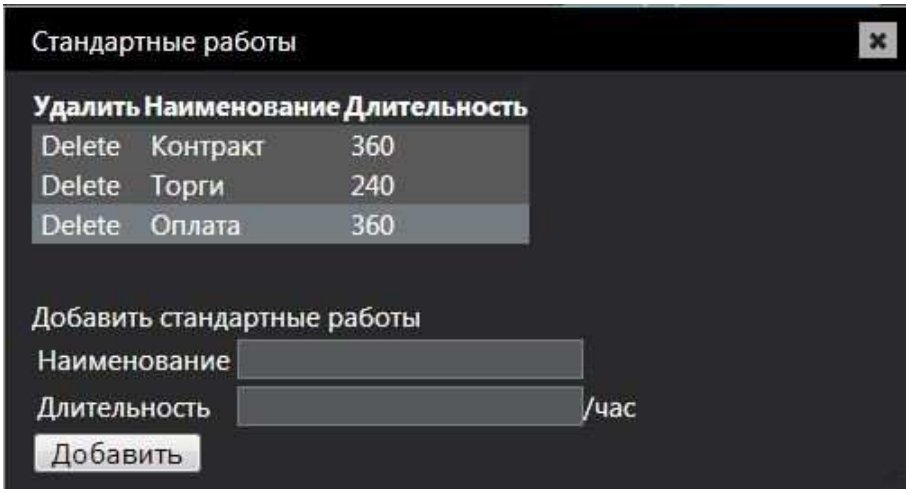

Рисунок 3.20 – Опции стандартных работ

3.3.3 Уровень Controller

Функционал контроллеров сводится к обеспечению возврата представления страниц, а также обновлению частичных представлений. Для этого используются методы типа ActionResult, принимающие параметры из асинхронных GET/POST-запросов, и возвращающие строго типизированные полные и частичные представления. Ниже представлен исходный код контроллера index.cshtml:

```
public class WorkController : Controller
 {
    public ActionResult Index()
    {
       return View();
    }
    [HttpGet]
    public ActionResult GetRootDetails(String Order, String Id)
```

```
        {
            var SelectedWork =
                AppContext.FindByOrder(Order).RootWorks.FirstOrDefault(x
=> x.Id == Id);
            var Model = new WorkViewModel(true);
            return PartialView("RootDetails", Model);
        }
        [HttpGet]
        public ActionResult GetDetails(String Order, String Id)
        {
            var SelectedWork =
                AppContext.FindByOrder(Order).RootWorks.FirstOrDefault(x
=> x.Id == Id);
            var Model = new WorkViewModel(false);
            return PartialView("Details", Model);
        }
    }
```

Как видно из листинга, основная задача контроллеров – правильно идентифицировать бизнес-объект, данные которого были переданы с уровня представления, сформировать для этого объекта модель представления нужного типа и отразить ее частичным представлением.

На рисунке 3.21 изображена диаграмма компонентов, которая отображает особенности физического представления системы. Диаграмма компонентов позволяет конкретизировать архитектуру разрабатываемой подсистемы, указав зависимости между программными компонентами, исполнять роль которых может исходных или исполняемый код.

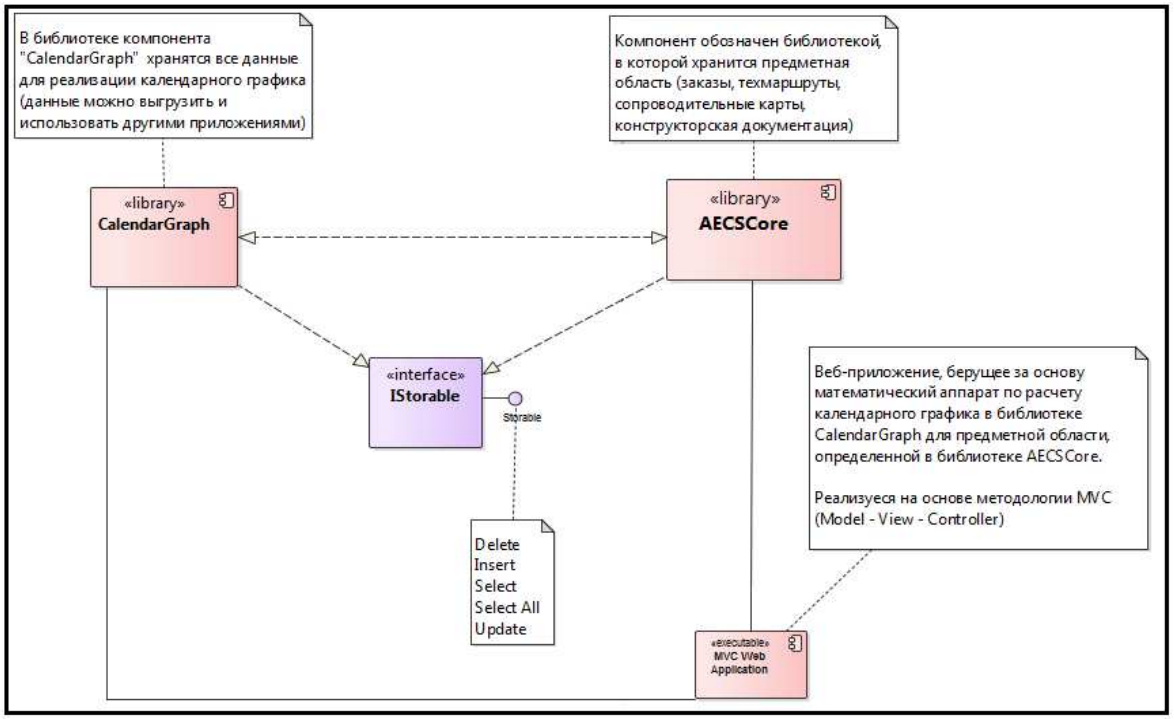

Рисунок 3.21 – Диаграмма компонентов

Диаграмма развертывания на рисунке 3.22 отображает общее представление об архитектуре проекта. Визуализация разрабатываемого объекта должна быть произведена в виде АРМ оператора, выполненная на языке JavaScript. Для описания бизнес-логики использован сервер

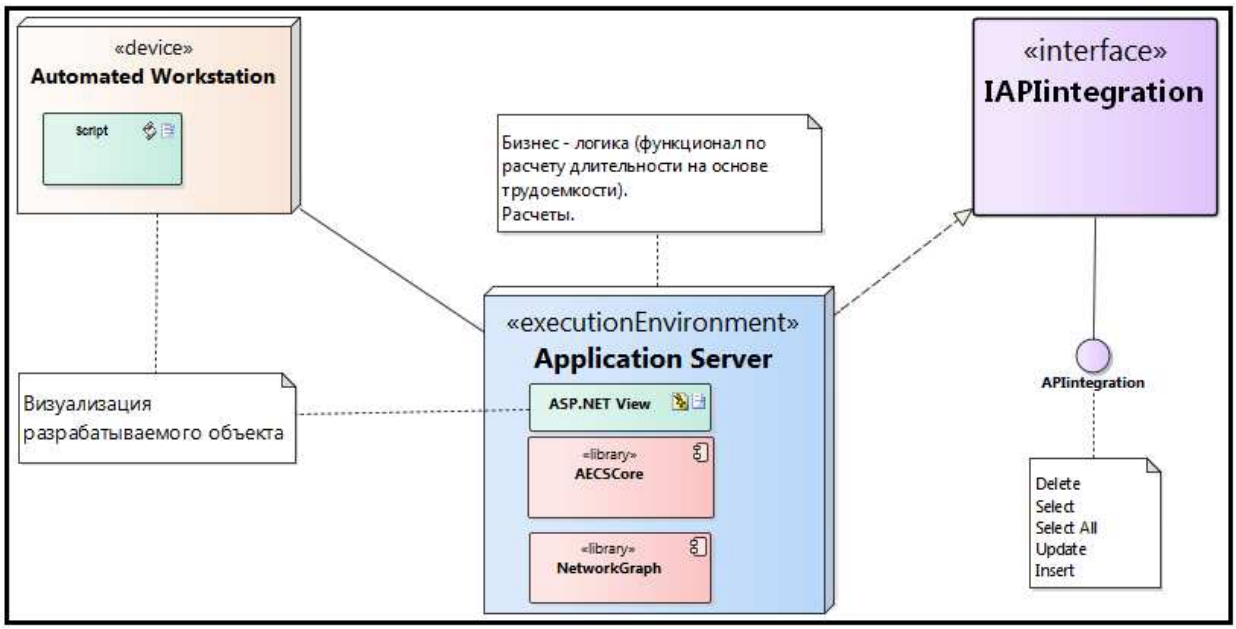

Рисунок 3.22 – Диаграмма развертывания

приложений, в котором AECScore – модуль предметной области предприятия; библиотека ASP.NetView – уровень представления данных; библиотека NetworkGraph реализует логику сетевого графика.

На рисунке 3.23 изображено построение календарного графика. В строке «номера заказа» вводится номер данного заказа. После чего, кликнув по кнопке «Построить», происходит построение календарного графика, на котором отображаются Стандартные и Основные работы.

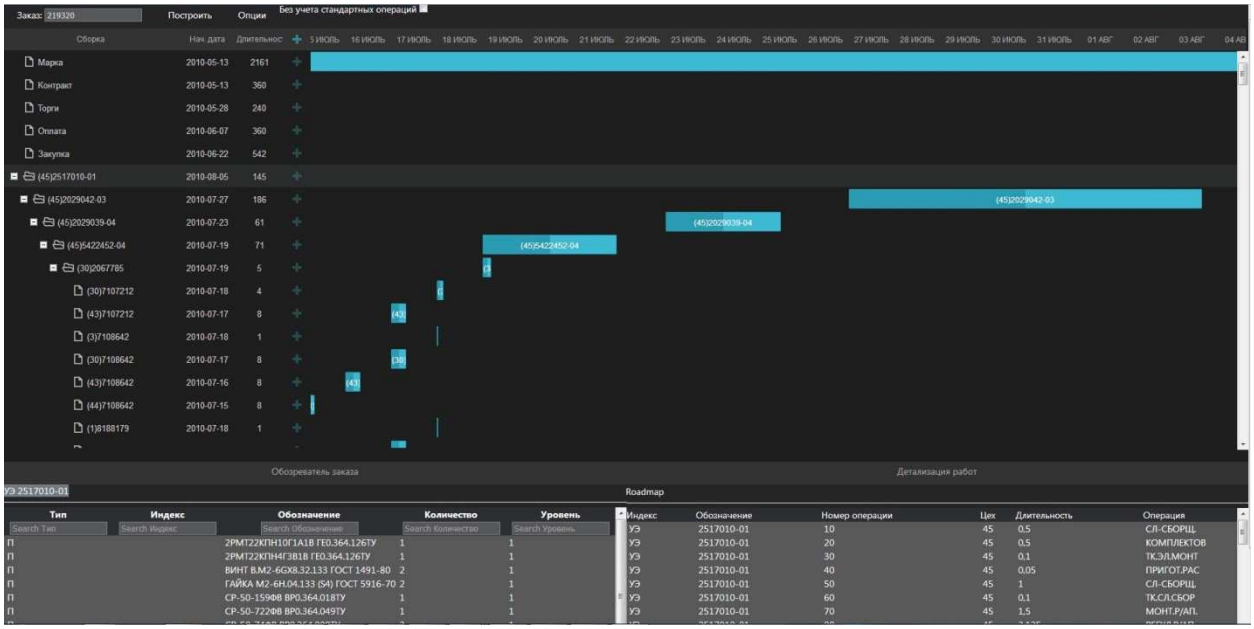

Рисунок 3.23 – Визуализированная диаграмма Ганта

3.4 Выводы по главе 3

В данном разделе диссертационного исследования описана разработка программного обеспечения автоматизированной системы производственного планирования, а именно:

– на основе анализа предметной области и объектной модели предоставленной аналитиком построена диаграмма классов, которая впоследствии развернута в программный код;

– построены диаграммы последовательностей, которые описывают функциональные требования к основным расчетным операциям;

– построены диаграммы реализации, включающие в себя диаграмму компонентов и диаграмму развертывания отображающие представление об архитектуре проекта;

– разработано веб-приложение с применением шаблона проектирования «Наблюдатель».

ЗАКЛЮЧЕНИЕ

В современных условиях производственное планирование и управления должно вестись с использованием соответствующих автоматизированных систем поддержки. Такого рода системы могут быть спроектированы и разработаны на основе известных или модифицированных методов сетевого планирования, методологий MES и APS. Предложенная система сетевого планирования предназначена как для одиночного применения, так и для использования в составе системы оперативного производственного планирования предприятия АО «НПП «Радиосвязь». Развитием или дополнением представленной разработки может являться оперативное управление и диспетчеризация оборудования, отслеживать его состояния, планирование загрузки с учетом ограничений по мощностям и ресурсам и т.д.

Система планирования спроектирована и разработана с применением CASE-инструментария. При реализации систем был осуществлен переход от аналитических абстракций и алгоритмов предметной области, к терминам объектно-ориентированной парадигмы – получены модели на языке UML, осуществлена их трансляция на язык C# с поддержкой механизма RTE для непрерывной синхронизации программного кода и модели.

Получена библиотека имплементации предметной области, пригодная для использования в других проектах отдела АСУП предприятия. С использованием технологии ASP.NET MVC разработано и развернуто на сервере приложений предприятия экспериментальное web-приложение для построения календарного графика, работающее с массивами отдела АСУП.

СПИСОК СОКРАЩЕНИЙ

В настоящей работе применены следующие сокращения:

АРМ – автоматизированное рабочее место;

АСУП – автоматизированные системы управления производством;

ОКП – оперативно-календарное планирование;

СУБД – системой управления базами данных;

ЕИП – единое информационное пространство;

БД – база данных;

ООП – объектно-ориентированное программирование.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Кривцов, А. М. Сетевое планирование и управление / А. М. Кривцов, В. В. Шеховцов. – Москва: Издательство «Экономика», 1978. – 191 с.

2 Дуболазов, В.А. Оперативно-календарное планирование на промышленном предприятии / В. А. Дуболазов. – Санкт-Петербург, 2000. – 36 с.

3 Коржов, К. Многоуровневые системы клиент-сервер / К. Коржов. – Москва: Издательство «Открытые системы», 1997. – 208 с.

4 Дейт, К. Дж. Введение в системы баз данных= Introduction to Database Systems / Дж. К. Дейт. – 8-е изд. – Москва: Вильямс, 2005. – 1328с. – ISBN 5-8459-0788-8(рус.) 0-321-19784-4 (англ.).

5 Фиайли, К. SQL: Руководство по изучению языка / К. Фиайли. – Москва: Peachpit Press, 2003. – 456 с.

6 Адамс, К. Администрирование сервера IIS 7 / К. Адамс. – Москва: Бином, 2010. – 362 с. – ISBN 978-5-9518-0367-2.

7 Исполнительные производственные системы [Электронный ресурс]. – Режим доступа: http://www.fobos– mes.ru/

8 Изучаем ASP.NET MVC 4 [Электронный ресурс]. – Режим доступа: http://metanit.com/sharp/mvc/index.php

9 Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. – Санкт-Петербург: Питер, 2015. – 320 с.

10 Маклафлин, Б. Изучаем Ajax / Б. Маклафлин. – Санкт-Петербург: Питер, 2007. – ISBN 978-5-91180-322-3

11 Хольцнер, С. Ajax Библия программиста = Ajax Bible / С. Хольцнер. – Москва: Диалектика, 2009. – С. 553. – ISBN 978-5-8459-1502-3

12 Брауде, Э. Технологии разработки программного обеспечения / Э. Брауде. – Санкт-Петербург: Питер, 2004. – 655 с.

13 Мацяшек Лешек, А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / А. Мацяшек Лешек: пер. с англ. – Москва: Вильямс, 2002. – 432 с.

14 Фримен, А. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов, 4-е издание Pro ASP.NET MVC 4, 4th edition / А. Фримен – Москва: Вильямс, 2013. – 688 с. – ISBN 978-5-8459-1867-3

15 Чедвик, Д. ASP.NET MVC 4. Разработка реальных веб-приложений с помощью ASP.NET MVC / Д. Чедвик, Т. Снайпер, Х. Панда: пер. с англ. – Москва: Вильямс, 2013. – 432 с. – ISBN 978-5-8459-1841-3

16 Интеграция связи [Электронный курс] – Режим доступа: http://www.itn.ru/solutions/system/

17 Чери, С. Логическое программирование и базы данных / С. Чери, Г. Готлоб, Л. Танка. – Москва: Мир, 1992. – 352 с.

18 Аппак, М. А. Автоматизированные рабочие места на основе персональных ЭВМ / М. А. Аппак. – Москва: Радио и связь, 1989. – 176 с.

19 Конструкторские документы и правила их оформления [Электронный курс] – Режим доступа: http://grafika.stu.ru/wolchin/umm/eskd/index.htm

20 Меняев, М. Ф. Информационные технологии управления: учебное пособие: В 3 кн.: Книга 3: Системы управления организацией / М. Ф. Меняев. – Москва: Омега-Л, 2003. – 464 с.

21 Тирон, Г. Г. Особенности стратегического и оперативного управления производством на предприятиях машиностроения / Г. Г. Тирон. – Пермь: НИИУМС, 2001. – 45с.

22 Гайдамакин, Н. А. Автоматизированные информационные системы, базы и банки данных. Вводный курс: учебное пособие / Н. А. Гайдамакин. – Москва: Гелиос АРВ, 2002. – 368 с.

23 Келасов, М.Ю. Основы проектирования производства / М. Ю. Келасов. – Санкт-Петербург: Петрополис, 2004г. – 231с.

24 Руководство по применению стандарта ИСО 9001:2000 при разработке программного обеспечения/ А.Л. Раскина: пер. с англ. – Москва: РИА «Стандарты и качество», 2002. – 104 с.

25 Подготовка ASP.NET 5 (Core) проекта и DNX окружения [Электронный курс] – Режим доступа: https://habrahabr.ru/post/278571/

26 Дейтел, Х. C# в подлиннике. Наиболее полное руководство / Х. Дейтел. – Санкт-Петербург: БХВ-Петербург, 2006. – 1056 с.

27 Уотсон, К. Visual C# 2010 Полный курс (Beginning Visual C# 2010) / К. Уотсон. – Москва: Диалектика, 2011. – 955 с.

28 Капулин, Д. В. Автоматизация планирования мелкосерийного производства сетевыми методами / Д. В. Капулин, М. В. Винниченко, Д. И. Винниченко // Прикладная информатика: научно-практический журнал / Московский финансово–промышленный университет «Синергия» – 2016. – Т. 11, № 6 (66). – С. 6–18.

29 Свид. 2016616446 Российская федерация. Государственная регистрационная программа для ЭВМ. Программный модуль разузлования материалокомплектов / Чорномаз Р. И., Капулин Д. В., Винниченко Д. И., Джиоева Н. Н., Котова М. В.; заявитель и правообладатель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ) (RU). – № 2016614024; заявл. 22.04.2016; опубл. 20.07.2016, Реестр программ для ЭВМ. – 1с.

30 Свид. 2017611504 Российская федерация. Государственная регистрационная программа для ЭВМ. Программный модуль построения объектов для диаграммы Ганта / Чорномаз Р. И., Капулин Д. В., Винниченко М. В., Винниченко Д. И.; заявитель и правообладатель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ) (RU). – № 2016663431; заявл. 08.12.2016; опубл. 06.02.2017, Реестр программ для ЭВМ. – 1с.

ПРИЛОЖЕНИЕ А

```
using System;
using System.Collections.Generic;
using System.Linq;
using Entities;

namespace Backend
{
    public class OrderService
    {
        public IList<GanttData> GetDetails(Order Order)
        {
            var Result = new List<GanttData>();
            var RootWorks = new List<GanttData>();

            // Корень заказа
            Order.StartTime = GetStartDateOrder(Order);

            var dbManager = DatabaseManager.GetInstance();
            var request = @"SELECT RootWorks.Duration, Zakaz.NaimZak
                                FROM(SELECT     SUM(rw.Duration)     AS
Duration, rw.id
                                  FROM dbo.RootWorks rw
                                  WHERE rw.id LIKE 'net%'
                                  GROUP BY rw.id) RootWorks
                                INNER   JOIN   dbo.tempPOSPRIMB   p   ON
RootWorks.id = p.id
                                INNER JOIN dbo.Zakaz ON Zakaz.Zakaz =
p.Z
                                GROUP       BY       RootWorks.Duration,
Zakaz.NaimZak";
            var Row = dbManager.SendRequest(request);
            var OrderDuration = Convert.ToInt32(Row[0]["Duration"]);
            Order.Name = Row[0]["NaimZak"].ToString();
            var Root = new GanttData()
            {
                id = 1,
                text = Order.Name,
                //order = 1, // не знаю что за параметр
                start_date  =  Order.StartTime.Date.ToString(),  //  Дата
открытия заказа, от нее будет считаться все остальное
                duration = OrderDuration,
                open = true,
                progress = 0
            };
            Result.Add(Root);

            ExplodeOrder(Order);
            RootWorks = GetRootWorks(Order);
```

```
            Result.AddRange(RootWorks);
            return Result;
        }

        //  Вызвать хранимую процедуру, которая разузлует заказ и
поместит результаты в таблицу tempPOSPRIMB
        public void ExplodeOrder(Order Order)
        {
            var dbManager = DatabaseManager.GetInstance();
            String             Request             =             @"EXEC
[1gb_x_t_mes].[dbo].[RAZUZLOVZAKAZ] 'netGraf',@OrderId";
            var RequestArgs = new Dictionary<String, Object>();
            RequestArgs.Add("OrderId", Order.Code);
            //  dbManager.SendCommand(Request,RequestArgs);
        }

        // Получить из базы корневые работы в виде: "Цех:Трудоемкость"
        public List<GanttData> GetRootWorks(Order order)
        {
            var Result = new List<GanttData>();
            var dbManager = DatabaseManager.GetInstance();
            String Request = @"Select    case when (a.tip='Б') then
a.id_record  else (ROW_NUMBER() over (order by a.id_record)) end as
id_record,  a.C,  a.TIP,  a.IND1,  a.PICH,  a.IND2,  a.P2NI,  a.Z,  a.id,
a.Parent,   CASE WHEN a.NV>8 THEN (nv/8)*24 ELSE a.NV END NV ,Depth
from
 (SELECT     tp.id_record,  tn.C,  tp.TIP,  tp.IND1,  tp.PICH,  tp.IND2,
tp.P2NI, tp.Z, tp.id, tp.Parent,   SUM( tn.NV) NV,Depth
FROM         tempPOSPRIMB AS tp
INNER JOIN TEXNORM AS tn ON tp.TIP = tn.TIP AND tp.IND1 = tn.IND AND
tp.PICH = tn.PICH
WHERE     (tp.id = 'netgraf')  AND ksz>0
GROUP  BY  tp.id_record,  tn.C,  tp.TIP,  tp.IND1,  tp.PICH,  tp.IND2,
tp.P2NI, tp.Z, tp.id, tp.Parent,Depth
) as a
order by depth DESC, CASE WHEN TIP='Б' THEN 1 WHEN TIP='Д' THEN 2 WHEN
TIP='Н' THEN 3 WHEN TIP='П' THEN 4 ELSE 5 end,pich,c";
            var Rows = dbManager.SendRequest(Request);
            //var Index = 2;
            var startDurationTime = order.StartTime.Date;
            order.StartTime = GetStartDateOrder(order);
            var  StandertWorkToGantt  =  GetStandartWorks().OrderBy(r  =>
r.id);

            foreach (var item in StandertWorkToGantt)
            {
                if (item.NameWork == "Контракт")
                    startDurationTime                                    =
startDurationTime.AddDays(Convert.ToInt16(item.Duration) * -1);
                var ganttStandartWorks = new GanttData
                {
```

```
                    id = Convert.ToInt32(item.id),
                    text = item.NameWork,
                    start_date = startDurationTime.ToString(),
                    open = false,
                    duration = Convert.ToDouble(item.Duration),
                    progress = 1,
                    parent=0
                };
                startDurationTime                                    =
startDurationTime.AddDays(Convert.ToInt16(item.Duration) / 24);
                Result.Add(ganttStandartWorks);
            }
            var                  EndDateStandartWork                 =
Convert.ToDateTime(Result.Last().start_date).AddDays(Result.Last().dur
ation/24);

            var countStandartWorks = Result.Count();
            foreach (var item in Rows)
            {
                if (item["TIP"].ToString() == "Б")
                {
                    var GanttData = new GanttData()
                    {
                        id = Convert.ToInt32(item["id_record"]),
                        text = "(" + item["C"] + ")" + item["PICH"],
                        start_date = startDurationTime.ToString(),
                        //order = 10,
                        open = true,
                        duration = Convert.ToDouble(item["NV"]),
                        parent                                       =
Convert.ToInt32(item["Parent"].ToString().Replace("-1", "0")),
                        progress = 0.5
                    };
                    Result.Add(GanttData);
                    startDurationTime                                =
startDurationTime.AddDays(Convert.ToDouble(item["Duration"])).AddDays(
1);
                }

            }
            var      orderResult      =      (Result.OrderBy(q      =>
q.start_date)).ToList();
            for (int index = orderResult.Count; index > 0; index--)
            {
                var item = Result[index - 1];
                if (item.parent != -1 && item.parent != 0)
                {
                    var parent = item.parent;
                    var    pos    =    Result.Where(q    =>    q.id    ==
parent).OrderByDescending(w => w.start_date).Select(q => q);
                    var dt = pos.ToList()[0].start_date;
```

```
                    var                         nv                          =
Math.Ceiling(Convert.ToDecimal(item.duration) / 24);

                    var  tempDate  =  ((Convert.ToDateTime(dt).AddDays(-
1)).AddDays(Convert.ToInt32(nv) * -1)).ToString();

                    item.start_date = tempDate;
                }
            }

            var prevPich = "";
            var actPich = "";
            DateTime lastdate = DateTime.Now;
            foreach (var item in Rows)
            {
                if (item["TIP"].ToString() == "Д")
                {
                    actPich = item["PICH"].ToString();
                    var parent = Convert.ToInt64(item["Parent"]);
                    var                         nv                          =
Math.Ceiling(Convert.ToDouble(item["NV"]));
                    var qwe = nv / 8;
                    var n = (int)qwe + 1;

                    var date = (Convert.ToDateTime(((Result.Where(w =>
w.id    ==    parent).Select(q    =>    q.start_date)).OrderBy(q    =>
q).Last()))).AddDays(-n);

                    var GanttData = new GanttData()
                    {
                        id = Convert.ToInt32(item["id_record"]),
                        text = "(" + item["C"] + ")" + item["PICH"],
                        start_date = date.ToString(),
                        //order = 10,
                        open = false,
                        duration = Convert.ToDouble(item["NV"]),
                        parent                                              =
Convert.ToInt32(item["Parent"].ToString().Replace("-1", "0")),
                        progress = 0.5
                    };
                    Result.Add(GanttData);

                    if (actPich == prevPich)
                    {
                        if (Result.Count() > countStandartWorks + 1)
                        {
                            var nDate = date.AddDays(-1);
                            date = nDate;

                            if (lastdate == date)
                            {
```

```
                                nDate = date.AddDays(-1);
                                date = nDate;
                                lastdate = date;
                            }

                            Result.Last().start_date= date.ToString();

                            lastdate = date;
                            var index = Result.Count() - 2;
                            Result[index].duration = 8;
                        }
                    }
                    prevPich = item["PICH"].ToString();
                }
            }
 var startdateWork =
                Result.Where(q  =>  Convert.ToDateTime(q.start_date)  >
EndDateStandartWork)
                    .OrderBy(w                                          =>
Convert.ToDateTime(w.start_date)).First();

            var  dif  =  (Convert.ToDateTime(startdateWork.start_date)  -
EndDateStandartWork).TotalHours;
            var buy = new GanttData
            {
                start_date = EndDateStandartWork.ToString(),
                id = 999999999,
                duration = dif-10,
                open = true,
                text = "Закупка",
                progress = 1,
            };

            var lst = (Result.Take(countStandartWorks));
            var NewResult = new List<GanttData>();
            NewResult.AddRange(lst);
            NewResult.Add(buy);
            NewResult.AddRange(Result.Skip(countStandartWorks+1));

            return  NewResult;
        }

        // Получить заказ по номеру
        public Order GetOrderById(Int32 OrderId)
        {
            var Result = new Order();

            var dbManager = DatabaseManager.GetInstance();
            String Request = @"SELECT zakaz, NaimZak
                               FROM [Zakaz]
                               WHERE zakaz = @Order";
```

```
            var RequestArgs = new Dictionary<String, Object>();
            RequestArgs.Add("Order", OrderId);
            var Rows = dbManager.SendRequest(Request, RequestArgs);

            if (Rows.Count > 0)
            {
                Result.Code = Rows[0]["zakaz"] as String;
                Result.Name = Rows[0]["NaimZak"] as String;
            }
            return Result;
        }

        public   List<Exploder>   GetExploderOrder(Int32   order,   int
parent)
        {
            var result = new List<Exploder>();
            var dbManager = DatabaseManager.GetInstance();
            String  Request  =  @"SELECT  id_record,  TIP,  IND1,  PICH,
Depth, KSZ
                            FROM[1gb_x_t_mes].dbo.tempPOSPRIMB    WHERE
Parent = @Parent and Z=@Z  and id like 'net%' and ksz > 0";
            var RequestArgs = new Dictionary<String, Object>();
            RequestArgs.Add("Parent", parent);
            RequestArgs.Add("Z", order);
            var Rows = dbManager.SendRequest(Request, RequestArgs);

            foreach (var Row in Rows)
            {
                var Exploder = new Exploder
                {
                    id_record = Row["id_record"].ToString(),
                    Type = Row["TIP"].ToString(),
                    Ind = Row["IND1"].ToString(),
                    Denotation = Row["PICH"].ToString(),
                    Depth = Row["Depth"].ToString(),
                    Amount = Row["KSZ"].ToString()
                };
                result.Add(Exploder);
            }

            return result;
        }

        public List<WorkDetail> GetWorkDetails(int idRecord)
        {
            var result = new List<WorkDetail>();

            var dbManager = new DatabaseManager();
            var Request = @"SELECT IND1, PICH, F, C, NV, NAZVO
                            FROM     [1gb_x_t_mes].dbo.BlocksByOperation
WHERE id = 'netGraf' AND id_record=@idRecord ";
```

```
            var RequestArgs = new Dictionary<String, Object>();
            RequestArgs.Add("idRecord", idRecord);
            var Rows = dbManager.SendRequest(Request, RequestArgs);

            foreach (var Row in Rows)
            {
                var WorkDetail = new WorkDetail
                {
                    Ind = Row["IND1"].ToString(),
                    Denotation = Row["PICH"].ToString(),
                    Chain = Row["F"].ToString(),
                    Department = Row["C"].ToString(),
                    Duration = Row["NV"].ToString(),
                    Operation = Row["NAZVO"].ToString()
                };
                result.Add(WorkDetail);
            }
            return result;

        }

        public DateTime GetStartDateOrder(Order order)
        {
            var dbManager = DatabaseManager.GetInstance();

            var request = @"SELECT TOP 1  do.DDoc DDoc
                    FROM TGInf t
                    INNER JOIN DocOsn do ON t.ID_DocOsn = do.ID_DocOsn
                                WHERE  Z  =   @order   ORDER  BY  t.NS
DESC";
            var RequestArgs = new Dictionary<string, object>();
            RequestArgs.Add("order", order.Code);
            var Rows = dbManager.SendRequest(request, RequestArgs);


            var                         result                          =
Convert.ToDateTime(Rows[0]["DDoc"].ToString());

            return result;
        }

        public List<StandartWorks> GetStandartWorks()
        {
            var result = new List<StandartWorks>();
            var dbManager = new DatabaseManager();
            var   request   =   @"SELECT   id,NameWork,   Duration   FROM
[1gb_x_t_mes].dbo.StandartWorks order by id";
            var rows = dbManager.SendRequest(request);

            foreach (var item in rows)
            {
```

```
                var StandartWorks = new StandartWorks
                {
                    id = item["id"].ToString(),
                    NameWork = item["NameWork"].ToString(),
                    Duration = item["Duration"].ToString()
                };
                result.Add(StandartWorks);
            }
            return result;
        }

        public void AddStandartWorks( string NameWork, string
Duration)
        {
            var dbManager = new DatabaseManager();
            var request = @" insert into
[1gb_x_t_mes].dbo.StandartWorks (NameWork, Duration) values ('" +
NameWork +
                        "','" + Duration + "' )";
            dbManager.SendRequest(request);
        }

        public void DeleteStandartWorks(int idWorks)
        {
            var dbManager= new DatabaseManager();
            var request = " delete from
[1gb_x_t_mes].dbo.StandartWorks  where id=" + idWorks;
            dbManager.SendRequest(request);
        }
    }
}
var tE, tW, RequestedOrder, stWork;

$(document).ready(function () {

    $('#ExploderTable thead tr:eq(1) th').each(function () {
        var title = $('#ExploderTable thead tr:eq(0)
th').eq($(this).index()).text();
        $(this).html('<input type="text" placeholder="Search ' + title
+ '" />');
    });

    tE = $('#ExploderTable').DataTable({
        "bPaginate": false,
        "bInfo": false,
        orderCellsTop: true,
        "columnDefs": [
             {
                 "className": "hidden",
                 "targets": [0]
             }
```

```
        ]
    });

    tE.columns().every(function (index) {
        $('#ExploderTable thead tr:eq(1) th:eq(' + index + ') input').on('keyup change', function () {
            tE.column($(this).parent().index() + ':visible')
                .search(this.value)
                .draw();
        });
    });

    $('.dataTables_filter').hide();

    tW = $('#WorkDetaleTable').dataTable({
        "bPaginate": false,
        "bInfo": false,
        "bFilter": false,
        "columns":
        [
            { "data": "Ind" },
            { "data": "Denotation" },
            { "data": "Chain" },
            { "data": "Department" },
            { "data": "Duration" },
            { "data": "Operation" }
        ],
        "order": [[2, "asc"]]
    });

    tE.on('click', 'tbody tr', function () {
        var field = $('td', this);

        var idRecord = field.eq(0).text();
        var ind = field.eq(2).text();
        var denotation = field.eq(3).text();

        var dv = $('#Navigate');
        dv.append("<span class='nv' style='background-color: #757c81; cursor:pointer' id=" + idRecord + ">" + ind + " " + denotation + "</span>      ");

        AjaxRequest("api/order/" + RequestedOrder + "/" + idRecord + "/Exploder", null, Exploder);
        AjaxRequest("api/Order/" + idRecord + "/WorkDetail", null, WorkDetail);

        $('.nv').on('click', function () {

            var item = $(this);
            idRecord = item.attr('id');
```

```
            AjaxRequest("api/order/" + RequestedOrder + "/" + idRecord
+ "/Exploder", null, Exploder);
            AjaxRequest("api/Order/" + idRecord + "/WorkDetail", null,
WorkDetail);
            for (var i = 0; i < 17; i++) {
                console.log(item.next('span').remove());
            }
        });
    });

    stWork = $('#stWork').dataTable({
        "bPaginate": false,
        "bInfo": false,
        "bFilter": false,

         "columnDefs": [
             {
                 "className": "hidden",
                 "targets": [0]
             }
        ]
    });


document.getElementById('addStandartWork').addEventListener('click',
addStandartWork);

    AjaxRequest("api/order/StandartWorks", null, AddStanartWork);
});



function Build() {
    RequestedOrder                                                    =
document.getElementById("RequestedOrderInput").value;
    //     AjaxRequest("api/order/"     +     RequestedOrder,     null,
ValidateOrder);
    AjaxRequest("api/order/" + RequestedOrder + "/-1" + "/Exploder",
null, Exploder);
    AjaxRequest("api/order/"   +   RequestedOrder   +   "/build",   null,
ShowDetails);

}

function Options() {
    $("#OptionsDialog").dialog({
        width: 500,
        // height: auto,
        modal: true
    });
}
```

```
function ValidateOrder(result) {
    //alert(result.Name);
}

function ShowDetails(result) {
    var data = JSON.stringify({ data: result });
    gantt.parse(data);
}

function Exploder(result) {
    tE.clear().draw();
    if (result.length !== 0) {
        for (var i = 0; i < result.length; i++) {
            tE.row.add([result[i].id_record,           result[i].Type,
result[i].Ind,         result[i].Denotation,         result[i].Amount,
result[i].Depth]).draw();
        }
    }
}

function WorkDetail(result) {
    tW.fnClearTable();
    if (result.length !== 0)
        tW.fnAddData(result);
}

function AddStanartWork(result) {

     stWork.fnClearTable();
    if (result.length !== 0) {
        for (var i = 0; i < result.length; i++) {
           // console.log(result[i]);

            stWork.fnAddData([result[i].id,                    "<span
onclick='DeleteStWork("   +   result[i].id   +   ")'   >Delete</span>",
result[i].NameWork, result[i].Duration]);
        }
    }


        // stWork.fnAddData(result);
}

function addStandartWork() {
    var naim = $("#stWorkNaim").val();
    var duration = $("#stWorkDuration").val();
    var match = duration.match(/^[0-9]+$/);
    console.log(match);
    if (naim !== "" && duration !== "" && Array.isArray(match) ===
true) {
```

```
        AjaxRequest("api/order/"   +   naim   +   "/"   +   duration   +
"/AddStandartWorks", null, function() {
            AjaxRequest("api/order/StandartWorks",                null,
AddStanartWork);
        });

    } else {
        alert("The data is not valid");
    }
}

function DeleteStWork(id) {
    if (confirm("Delete?")) {
        AjaxRequest("api/order/" + id + "/DeleteStandartWorks", null,
function() {
            AjaxRequest("api/order/StandartWorks",                null,
AddStanartWork);
        });
    }
}
$(document).ready(function () {
    var ContainerId = "gantt_here";

    gantt.config.duration_unit = "hour";
    gantt.config.time_step = 7;
    gantt.init(ContainerId);
    gantt.locale = {
        date: {
            month_full: ["Январь", "Февраль", "Март", "Апрель", "Мая",
"Июнь", "Июль",
            "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"],
            month_short: ["Янв", "Фев", "Март", "Апр", "Май", "Июнь",
"Июль", "Авг", "Сент",
            "Окт", "Нояб", "Дек"],
            day_full:   ["Воскресенье",   "Понедельник",   "Вторник",
"Среда", "Четверг", "Пятница",
            "Суббота"],
            day_short: ["Вс", "Пн", "Вт", "Ср", "Чт", "Пт", "Сб"]
        },
        labels: {
            new_task: "New",
            icon_save: "Save",
            icon_cancel: "Cansel",
            icon_details: "Детали",
            icon_edit: "Изменить",
            icon_delete: "Удалить",
            confirm_closing: "",
            confirm_deleting: "Task will be deleted permanently, are
you sure?",

            section_description: "Описание",
```

```
            section_time: "Период",
            column_duration: "Длительность",
            column_start_date: "Нач. дата",
            column_text: "Сборка",
            /* link confirmation */

            confirm_link_deleting:   "Dependency   will   be   deleted
permanently, are you sure?",
            link_from: "From",
            link_to: "To",
            link_start: "Start",
            link_end: "End",

            minutes: "Минуты",
            hours: "Часы",
            days: "Дни",
            weeks: "Неделя",
            months: "Месяц",
            years: "Год"
        }
    };

    function EnableScroller() {
        // TODO : В плагин JQuery mousewheel вручную включил 'event
capture', подумать над альтернативой
        $('body').mousewheel(function (event, delta) {
            gantt.scrollTo(gantt.getScrollState().x - delta * 100);
            //gantt.scrollTo(gantt.getScrollState().y - delta * 100);
            event.preventDefault();
        });
    }

    EnableScroller();
});

//@Depricated
function GetSampleData() {

}
```

ПРИЛОЖЕНИЕ Б Акт об использовании

Акционерное общество
«Научно-производственное предприятие «Радиосвязь»
(АО «НПП «Радиосвязь»)

ул. Декабристов, д. 19, Красноярск, 660021
Тел. (391) 221-22-78, тел./факс (391) 221-62-56, 221-79-30 E-mail:kniirs1@mail.kts.ru
ОКПО 44589548, ОГРН 1122468072231, ИНН/КПП 2460243408/246750001

Исх. №_____ от «_____»_____20____ г.
На № _____ от «_____»_____20____г.

АКТ
об использовании результатов научно-исследовательской работы
Винниченко Марины Валерьевны, Винниченко Дмитрия Игоревича,
Чорномаз Руслана Игоревича на предприятии АО «НПП «Радиосвязь»

Настоящий акт составлен комиссией в составе:
Председатель – Казанцев М. А., начальник отдела АСУП
Члены комиссии:
Дементьева Н. А., зам. начальника отдела АСУП
Гарифулин Э. Ф., начальник производственного отдела

Комиссия в вышеназванном составе составила настоящий акт о том, что в АО «НПП «Радиосвязь» в деятельности производственных подразделений использованы результаты научно-исследовательской работы «Разработка интегрированного решения информационной поддержки процессов производственного планирования и управления», полученные в рамках выполнения магистерских диссертаций авторами Винниченко М. В., Винниченко Д. И., Чорномаз Р. И.

В качестве основного результата следует отметить разработанную автоматизированную систему, входящую в состав комплекса задач по производственному планированию, применение которой позволило автоматизировать процесс построения сетевого графика для организации деятельности производственных подразделений. Использование разработанного программного продукта позволяет однозначно определять сроки запуска/выпуска производственного заказа, планировать выполнение цепочек работ, выявлять причины возникновения внеплановых задержек.

В качестве замечания можно отметить, что применение данного комплекса возможно только для серийных изделий, прошедших полную технологическую подготовку, и не применимо для планирования опытных образцов.

Председатель: Казанцев М. А.
Члены комиссии: Дементьева Н. А.
Гарифулин Э. Ф.

ПРИЛОЖЕНИЕ В Свидетельство о государственной регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

RU **2017611504**

(12) **ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ**

Номер регистрации (свидетельства):
2017611504

Дата регистрации: **06.02.2017**

Номер и дата поступления заявки:
2016663431 08.12.2016

Дата публикации: 06.02.2017

Авторы:
Чорномаз Руслан Игоревич (RU),
Капулин Денис Владимирович (RU),
Винниченко Марина Валерьевна (RU),
Винниченко Дмитрий Игоревич (RU)

Правообладатель:
Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ) (RU)

Название программы для ЭВМ:
Программный модуль построения объектов для диаграммы Ганта

Реферат:
Программа предназначена для использования в качестве отдельной программы в среде с поддержкой ASP.NET Core 1.0. Программа реализует функционал по определению соответствия между базой данных и объектами приложения. Для визуализации диаграммы Ганта выполняется инициализация и передача коллекции объектов клиенту через Web Api. Программа может быть использована как отдельное программное приложение, так и в составе системы производственного планирования.

Тип реализующей ЭВМ: IBM PC-совмест. ПК на базе процессора Intel Pentium III

Язык программирования: C#

Вид и версия операционной системы: Windows XP и выше

Объем программы для ЭВМ: 20 Кб

ПРИЛОЖЕНИЕ Г Публикация

прикладная
ИНФОРМАТИК@
научно-практический журнал

Том 11. №6 (66). 2016
Ноябрь – декабрь

ISSN 1993-8314

Университет «Синергия»

Журнал включен в Перечень ведущих периодических изданий, рекомендованных ВАК для публикации результатов диссертационных исследований.

Журнал выходит с 2006 г. Периодичность издания — 6 раз в год.

Журнал индексируется в российских и зарубежных базах научной периодики eLIBRARY (РИНЦ), Russian Science Citation Index (RSCI) на платформе Web of Science, ВИНИТИ, Ulrich's Periodicals Directory

Учредитель и издатель: Негосударственное образовательное частное учреждение высшего образования «Московский финансово-промышленный университет «Синергия»

Адрес редакции и издателя:
129090, Москва, ул. Мещанская, д. 9/14, стр.1 (юрид.)
125190, Москва, Ленинградский просп., д. 80, корп. Г, офис 612 (4)
Тел.: +7 (495) 987-43-74 (доб. 33-04)
e-mail: edit@synergy.ru; www.appliedinformatics.ru

Продолжение приложения Г

Окончание приложения Г

Д. В. Капулин, канд. техн. наук, доцент, Сибирский федеральный университет, г. Красноярск, dkapulin@sfu-kras.ru

М. В. Винниченко, магистрант, Сибирский федеральный университет, г. Красноярск, marina91293@mail.ru

Д. И. Винниченко, магистрант, Сибирский федеральный университет, г. Красноярск, dima_2504@mail.ru

Автоматизация планирования мелкосерийного производства сетевыми методами

В работе рассматриваются особенности сетевого планирования и управления мелкосерийным дискретным производством. Сформулированы требования к составу информационного обеспечения для автоматизации процесса построения календарного графика на основе сетевых моделей и требования к программному продукту, учитывающему особенности радиоэлектронного мелкосерийного производства. Предложена программная архитектура и проведено опробование системы автоматизированного планирования мелкосерийного производства электронной аппаратуры. Эта система позволяет осуществлять динамическую корректировку производственного плана.

Ключевые слова: сетевое планирование и управление, производственное планирование, календарный график, сетевой график, разузлование.

Введение

В процессах производственного планирования сетевые методы широко распространены. Они позволяют поднять качество и координацию действий в случаях, когда достижение целей планирования зависит от многих факторов, связанных с получением и переработкой информации, рациональным распределением ресурсов, построением взаимоотношений смежных подразделений. Использование алгоритмов сетевого планирования на предприятиях также позволяет провести визуализацию сложных процессов управления, повысить их эффективность, осуществить разностороннее исследование системы управления проектами и организацией в целом.

Основной результат сетевого планирования, как и иных видов планирования, — календарный план, в котором комплексные производственные задания разбиваются на отдельные, расположенные в технологической и временной последовательности, работы. Но при этом в отличие от методов линейного планирования в ходе выполнения процессов сетевого планирования оптимизации подвергается значительное число параметров, что способствует сокращению длительности выполнения проектных и производственных работ, снижению финансовых издержек за счет высокой координации деятельности различных подразделений предприятия.

Особое значение модели сетевого планирования приобретают на дискретных сборочных производствах, отличающихся мелкой серией. На таких предприятиях затруднительно применять линейные модели планирования из-за значительных сложностей при их динамической корректировке в случае из-

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Системы автоматики, автоматизированное управление и проектирование

УТВЕРЖДАЮ
Заведующий кафедрой
_____С. В. Ченцов
«22» 06. 2017г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПРОИЗВОДСТВЕННОГО ПЛАНИРОВАНИЯ

Направление 27.04.04 Управление в технических системах
Магистерская программа 27.04.04.01 Интегрированные системы управления производством

Научный руководитель	_____	22.06.2017 г.	доцент, канд. техн. наук Д. В. Капулин
Выпускник	_____	22.06.2017 г.	Д. И. Винниченко
Рецензент	_____	22.06.2017 г.	зав. каф., канд. техн. наук А. С. Кузнецов
Нормоконтролер	_____	22.06.2017 г.	Т. А. Грудинова

Красноярск 2017