

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ	5
ВВЕДЕНИЕ	7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ РАЗРАБОТКИ, СРАВНЕНИЕ С АНАЛОГАМИ	10
1.1 Описание предметной области с анализом существующих ре- шений и их недостатков исходя из целей работы и критериев сравнения, которые должны соответствовать искомым реше- ниям поставленных задач.	10
1.1.1 МАС	10
1.1.2 Исследование операций	12
1.2 Разработка возможных направлений проведения исследова- ний и методов решений отдельных задач	13
1.3 Обоснование применяемых технологий и инструментов. . . .	14
1.4 Уточненная постановка задачи и требования к прототипу ре- шения.	17
1.5 Выводы	19
2 ПРОЕКТИРОВАНИЕ	20
2.1 Проектирование архитектуры прототипа системы решения, включая разработку структуры (компоненты и связи между ними), внутреннего и внешнего функционирования.	20
2.1.1 Анализ доступных средств платформы .Net	20
2.1.2 Определение алгоритмов, по которым будет работать программный модуль	21
2.1.3 Выбор типа базы данных и её провайдера.	22
2.1.4 Проектирование архитектуры	23
2.2 Разработка прототипов алгоритмов, структур данных.	25
2.2.1 Типы данных Google OR-Tools	25

2.2.2	Определение требований к модели	28
3	РЕАЛИЗАЦИЯ ПРЕДЛАГАЕМОГО РЕШЕНИЯ И ЕГО ОЦЕНКА .	29
3.1	Реализация прототипа технического решения	29
3.1.1	Описание инструментов и форматов данных	29
3.1.2	Описание модели базы данных	30
3.1.3	Прототипирование ЕАМ-системы	32
3.1.4	Моделирование системы средствами программирова- ния в ограничениях	37
3.2	Экспериментальные исследования прототипа решения	45
	ЗАКЛЮЧЕНИЕ	48
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ (СТАРЫЙ)	49

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

ПО – программное обеспечение.

EAM – Enterprise Asset Management - система управления активами.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Makespan – период времени, который проходит от начала работы до ее завершения.

Job-shop – оптимизационная проблема в информатике и исследовании операций. Это вариант оптимального планирования заданий. В общей проблеме планирования заданий нам дается n заданий J_1, J_2, \dots, J_n с различным временем обработки, которые должны быть запланированы на m машинах с различной вычислительной мощностью, при этом мы пытаемся минимизировать время выполнения (makespan) - общую длину расписания.

SQL – structured query language - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

.NET – это бесплатная, кроссплатформенная платформа для разработчиков с открытым исходным кодом для создания различных типов приложений.

ORM – Object-Relational Mapping - объектно-реляционное отображение, технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

ADO.NET – ActiveX Data Object для .NET - технология, предоставляющая доступ к данным и управление ими, хранящимися в базе данных или других источниках.

Entity Framework – ORM фреймворк с открытым исходным кодом для ADO.NET.

LINQ – встроенный в .Net язык запросов к данным.

ВВЕДЕНИЕ

ЕАМ-системы - чрезвычайно важная часть производственного процесса предприятий. Это инструмент управления производительностью, рисками и затратами, связанными с физическими (производственными) активами организаций.

Для современных производственных процессов актуальными являются задачи оптимизации: оптимизация маршрутов логистики, рабочего времени, производственных затрат и т. д. Масштаб таких оптимизационных задач может быть непосилен для решения сугубо человеческими ресурсами, даже при использовании ЕАМ-систем, поэтому необходимо включать в область решения данных задач программно-аппаратные комплексы, решающие подобные задачи.

С 2011 года мировыми конгломератами ставится вопрос о четвертой промышленной революции - индустрии 4.0. Отходя от её утопических идей, индустрия 4.0 это гигантский шаг в сторону модернизации всех аспектов производства.

Большие данные, интернет вещей, блокчейн, искусственный интеллект, цифровые двойники - все эти технологии, которые уже нашли те или иные способы применения в различных областях науки и техники, только и ждут своего часа применительно к производству и производственным процессам. Уже сейчас самые передовые производства невероятно трудно представить без всех этих технологий.

Так новый передовой завод Intel [1] - полностью автоматизированное цифровое производство, где все самые важные и ответственные роли исполняют именно роботизированные механизмы или автономные дроны с интеллектом роя.

На текущий момент на рынке представлено достаточно много решений, связанных с ЕАМ и ERP системами, как отечественные:

- 1С ERP

- "TRIM"
- "Галактика ERP"

Поменял местами трим и 1с

так и зарубежные:

- SAP ERP
- IBM Maximo

Все представленные EAM-системы (как отдельные, так и в составе ERP) позволяют в полуавтоматическом режиме планировать работы, то есть указать какое количество ресурсов и когда понадобится для обслуживания тех или иных активов.

Во всех приведённых выше системах данное планирование происходит по потребности, то есть в долгосрочной перспективе не учитывает внезапные изменения, которые могут произойти с ресурсами, и их всевозможные коллизии - ситуации, когда в один момент времени один и тот же ресурс может быть задействован в разных работах.

Исходя из этого можно сказать, что в таких системах стоит до сих пор нерешенная проблема актуализации ресурсного планирования. Именно решение этой проблемы является целью данной работы.

В качестве объекта работы выступают данные, полученные из системы класса EAM, структурированные определенным образом. В общем виде - база данных с набором сущностей вида "работа", "тип работы", "исполнитель" и так далее.

Цель данной работы - создание открытого мультиплатформенного решения, предназначенного для ресурсного планирования в системах управления физическими активами.

В данной работе решается задача ресурсного планирования, то есть распределение заданий по исполнителям в заданном временном интервале с учетом ограниченного количества имеющихся ресурсов.

В результате работы планируется получить готовое программно-техническое решение, которое может взаимодействовать с имеющимися

ЕАМ-системами и обеспечивать для них решение задачи ресурсного планирования с заданными граничными условиями.

В процессе работы был проведен анализ доступных средств и методов в области планирования ресурсов, была составлена модель, отражающая в общем виде структуру ресурсов, для которой необходимо проводить процедуры планирования, а также проведен эксперимент, показывающий соответствие модели поставленной задаче.

Работа состоит из Введения, N глав и Заключения, содержит список литературных источников из M наименований и Приложения/ий.

1 Анализ предметной области разработки, сравнение с аналогами

1.1 Описание предметной области с анализом существующих решений и их недостатков исходя из целей работы и критериев сравнения, которые должны соответствовать искомому решению поставленных задач.

Одной из областей в операционных исследованиях является составление и оптимизация расписаний. Когда количество работ, человеческих ресурсов и различных механизмов невелико, то составление расписания вполне посильная для человека задача. Но когда количество объектов, учитываемых при планировании, возрастает, например, до уровня предприятия, то задача перестаёт быть тривиальной даже для целого отдела планирования. В таких случаях становится необходимым использование средств автоматизации планирования, примером которых являются мультиагентные системы или программные комплексы комбинаторной оптимизации - отдельные или в рамках ЕАМ-систем.

В целом можно разделить существующие методы построения и оптимизации подобных моделей на два основных подтипа - Мультиагентные системы (МАС) и Исследование операций (Operations research - далее OR) в рамках целочисленного программирования (программирования в ограничениях).

1.1.1 МАС

Мультиагентная система – система, состоящая из множества взаимодействующих друг с другом агентов. Каждый агент в такой системе независим и не имеет представления о системе в целом, но знает свои задачи и потребности, а также задачи и потребности агентов на одном с ним уровне. Для простоты понимания можно условно сказать, что агенты представляют собой

нечто схожее с NPC (от англ. Non-Player Character - Неигровой персонаж) в компьютерных играх.

Перевод с английского

Мультиагентные системы начали широко развиваться в девяностых годах и достигли своего пика к середине двухтысячных годов, после чего парадигма создания агентных сетей как чего-то, требующего отдельных подходов умерла в связи развитием и упрощением программирования в целом.

Согласно статьи Virginia Dignum и Frank Dignum [2], несмотря на существование целых организаций по изучению мультиагентных систем, ни за время существования этих организаций, ни за время их работы, МАС не смогли проявить себя ни в одной из заявленных областей, кроме как моделирования поведения персонажей в компьютерных играх [3]. Исследователи связывают неудачу МАС со слабой способностью конкурировать с другими компьютерными технологиями того времени, в первую очередь вызванной тем, что из-под пера ассоциации или заинтересованных исследователей так и не вышло платформы МАС, которую могли бы использовать все желающие за пределами «закрытого» сообщества. Второй проблемой при попытке применить исследования агентов на практике, по их мнению, стала неспособность охватить или учесть существенные характеристики проблемы, для решения которой используются агенты. Аналогичные выводы были представлены Городецким и Скобелевым в статье 2017 года [4].

Убрал название статьи

В целом всё-таки можно сказать, что МАС – достаточно перспективное направление, второе дыхание которому могут дать современные разработки в области искусственного интеллекта, нейросетей и программирования.

Главным плюсом (из которого в общих случаях следует и главный минус) мультиагентных систем является скорость реакции на изменения в системе и внешние факторы. Предполагается, что такая система фактически моментально должна реагировать на «происшествия» и сразу же выдавать удовлетворяющее решение с приемлемым качеством. Данное свойство мульт-

тиагентных систем делает их максимально пригодными для использования в качестве систем решения задач реального времени или в качестве модели-двойника.

Под моделью-двойником представляется виртуальный клон реальной системы, где все объекты взаимодействия являются агентами. Предполагается, что такие модели будут использоваться, например, для симулирования внештатных ситуаций в системе и оценки реагирования системы и агентов на эти ситуации.

Так как МАС предлагают в целом «приемлемые решения», то в областях, где важна каждая мелочь и требуется наиболее полный точный результат, такой подход не подойдет. МАС вполне можно использовать на этапах конкретной эксплуатации, где как раз необходимы оперативные принятия решений, а вот в задаче установки оптимальных начальных условий такой подход скорее всего будет неприменим.

1.1.2 Исследование операций

Для получения наиболее точных и оптимальных решений применяется дисциплина под названием «Исследование операций». Согласно Википедии [5], «ИО - дисциплина, занимающаяся разработкой и применением методов нахождения оптимальных решений на основе математического моделирования, статистического моделирования и различных эвристических подходов в различных областях человеческой деятельности. Иногда используется название «математические методы исследования операций».

Методы исследования операций являются основными в решениях таких задач как планирование, транспортные потоки, маршрутизация, упаковка и задачи о назначениях. В варианте, представленном в статье, предлагается использовать методы комбинаторной оптимизации, а именно – целочисленное программирование в ограничениях.

Программирование в ограничениях — это теория решения комбинаторных задач, опирающаяся на большое количество знаний из областей искус-

ственного интеллекта, информатики и исследования операций. В программировании в ограничениях декларативно задаются ограничения для набора переменных входных данных. Ограничения не определяют последовательность действий для получения искомого результата, а указывают свойства и особенности для нахождения конечного результата.

Оптимизация с ограничениями, или программирование с ограничениями (англ.- constraint programming, далее CP), — это название, данное идентификации выполнимых решений из очень большого набора возможных решений. CP основывается на выполнимости (нахождении выполнимого решения), а не на оптимизации (нахождении оптимального решения) и фокусируется на ограничениях и переменных, а не на целевой функции. Фактически, задача CP может даже не иметь целевой функции - цель может заключаться просто в том, чтобы сузить большое множество возможных решений до более управляемого подмножества путем добавления ограничений к задаче.

1.2 Разработка возможных направлений проведения исследований и методов решений отдельных задач

Подкорректировал название

Рассказать про другой опенсорс, про другие программные методы реализации, еще какой метод.

Исходя из требований следует, что разработанная система будет предназначена для перепланирования некоторого множества предварительно заданных в ЕАМ-системе работ.

На текущий момент основным средством ресурсного планирования является фактически ручной метод распределения работ, при котором ответственные люди вручную указывают потребность в работах и ресурсах. При больших объемах работ при таком типе планирования теряется одна из важных потребностей планирования, а именно потребность в актуальности расписания.

Когда количество работ велико, становится физически невозможно одними человеческими силами отследить или как-то исправить пересечения в потребности тех или ресурсов, так как таких коллизий может быть сотни или тысячи даже на промежутке в неделю.

Решение данной проблемы программными средствами тоже сама по себе нетривиальная задача, так как в таком случае на систему необходимо смотреть в целом, а не разрешать отдельные коллизии, а каких-либо общепринятых адекватных решений для данной проблемы до сих пор представлено не было.

Исходя из вышесказанного одним из главных направлений проведения исследований в данной области будет анализ доступных средств и инструментов, подходящих под задачи ресурсного планирования, и вследствие исследование конкретных методов решения поставленных задач уже у выбранного направления.

1.3 Обоснование применяемых технологий и инструментов.

Реализация различных методов комбинаторной оптимизации представляется достаточно трудоемкой задачей, поэтому предполагается использование готовых библиотек и фреймворков, реализующих методы исследования операций.

Одними из самых популярных фреймворков для решения задач исследования операций являются открытые GLPK [6], LP_Solve [7], MiniZinc [8], Google OR-Tools [9], CHOCO [10] и проприетарные IBM CPLEX [11] и Gurobi [12].

К дальнейшему рассмотрению предлагается три из приведенных выше решений – Открытый и бесплатный Google OR-Tools, как целевой фреймворк в данной работе, и проприетарные IBM CPLEX и Gurobi, как лидирующие решения в отрасли.

Обоснование выбранных для сравнения фреймворков

Приведем частичные результаты бенчмарков между Google OR-Tools и IBM CPLEX, полученные в статье "Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem" [13], в таблице 1.1.

Таблица 1.1 – Сравнение Or-Tools и CPLEX в задаче Job-shop (меньше - лучше)

Номер теста	Одно ядро		Четыре ядра	
	CPLEX msp (sec)	OR-Tools msp (sec)	CPLEX msp (sec)	OR-Tools msp (sec)
1	1234 (1.9)	1234 (1.8)	1234 (3.3)	1234 (1.6)
2	943 (0.7)	943 (0.7)	943 (1.4)	943 (0.4)
3	656 (1169.3)	660	565 (525)	661 (1200)
4	682	679	680	679
5	685	695	694	689
6	55 (0)	55 (0)	55 (0)	55 (0)
7	930 (3.8)	930 (5)	930 (5.9)	930 (2.9)
8	1165 (1.4)	1165 (5)	1165 (0.5)	1165 (3.4)
9	666 (0)	666 (0.1)	666 (0)	666 (0.1)
10	655 (0.3)	655 (0.3)	655 (0.5)	655 (0.1)

В тестах производительности между Google OR-Tools и IBM CPLEX стоял вопрос решения проблемы составления расписания работы в цехе (англ. Job-shop scheduling problem), которая приобрела особую известность благодаря своей простой формулировке, приводящей к трудно решаемым задачам.

Наиболее типичным критерием оптимизации является минимизация промежутка времени (англ. makespan), т.е. промежутка времени между началом первой операции и окончанием последней. Проблема представляется в виде набора заданий, которые должны быть обработаны набором машин. Каждое задание - это последовательность операций, каждая операция должна быть обработана определенной машиной и занимает определенное время обработки. Каждое задание имеет определенный порядок операций, который должен соблюдаться. Допустимым решением этой задачи является такая последовательность операций на каждой машине, при которой нет перекрытия

времени между двумя операциями на одной машине и соблюдается порядок операций.

Как видим из статьи, по результатам тестов на момент 2017 года фреймворк IBM CPLEX в целом чаще выигрывал у OR-Tools в размере makespan и скорости получения решения, при этом фреймворк OR-Tools показывал себя лучше в многоядерных тестах большого масштаба.

В другой статье [14] представлены уже результаты сравнения коммерческих IBM CPLEX и Gurobi. По результатам тестов последние версии обоих решателей в целом идут нога в ногу и выдают результаты очень близкие как по качеству, так и по скорости нахождения решений.

По результатам бенчмарков можно сказать, что между фреймворками Google OR-Tools, IBM CPLEX и Gurobi (особенно в многоядерной конфигурации) существует условный паритет по скорости и качеству решения оптимизационных задач, и в целом предугадать, кто окажется впереди в условиях реальной задачи невозможно.

Одним из наиболее доступных фреймворков является Google Or-Tools. OR-Tools - это программное обеспечение с открытым исходным кодом для комбинаторной оптимизации, которая направлена на поиск наилучшего решения проблемы из очень большого набора возможных решений. Последние несколько лет именно OR-Tools среди всех открытых библиотек занимает первые места в соревнованиях MiniZinc Challenge по программированию в ограничениях [15]. Также, в отличие от большинства остальных фреймворков имеет большее количество официально написанных интерфейсов под разные языки программирования – C++ (изначально написан именно на нём), C#, Java, Python.

Фреймворк Google OR-Tools поддерживает решатели под задачи линейной оптимизации (решения проблемы, смоделированной как набор линейных зависимостей.), целочисленной оптимизации (Смешанное целочисленное программирование - Mixed Integer Programming – MIP; некоторые переменные обязательно представляют собой целые числа) и программирование в ограничениях. Все эти решатели могут быть использованы для

решения таких задач, как: решение sudoku, диета Стиглера [16], шахматные задачи, планирование, составление оптимальных маршрутов и т.д.

В результате обзора и анализа методов решения поставленной задачи, на данный момент наиболее подходящим методом можно признать метод комбинаторной оптимизации и программирования в ограничениях, так как он явно предоставляет набор устоявшихся практик и решений, показавших себя во множестве проектов.

Основным средством предлагается использовать фреймворк Google OR-Tools, решающий задачи комбинаторной оптимизации, так как данный фреймворк является бесплатным, открытым, может быть использован в коммерческой разработке и покрывает большую часть областей и задач исследования операций.

Мультиагентные системы на данный момент, напротив, по результатам анализа показали себя непригодными к повсеместному использованию, так как не предоставляют доступных средств для реализации и не были широко протестированы в реальных условиях.

1.4 Уточненная постановка задачи и требования к прототипу решения.

Имеются следующие начальные условия:

- Есть множество запланированных работ (ЗР) на заданном временном интервале. Каждая работа имеет расчетные плановую дату/время начала выполнения и плановую дату/время завершения выполнения (разница между этими датами определяет длительность работы).
- Каждая работа имеет перечень исполнителей и механизмов, необходимых для ее выполнения (каждый из этих перечней может быть пустым). Для каждого исполнителя и механизма задана потребность, выраженная в чел.*час. (либо, аналогично, машино*час.). В качестве исполнителя (механизма) может быть указан либо конкретный исполнитель (штатная единица (ШЕ) – для человека или конкретный ме-

ханизм), либо требуемая квалификация (профессия) исполнителя или необходимый тип механизма.

- Имеется подразделение (цех, участок и т.п.), располагающее определенными людскими ресурсами и определенным набором механизмов. Состав людских ресурсов определяется штатным расписанием (оно имеет структуру), перечень доступных механизмов также задан в этой структуре. Каждая ШЕ (людской ресурс) в составе штатного расписания имеет свою профессию, а механизм – соответствующий ему тип механизма.

Задача:

На основании имеющихся исходных данных постараться построить такой порядок выполнения работ, чтобы в каждый день суммарная потребность в людских ресурсах и механизмах на выполнение всех работ не превышала объем располагаемых ресурсов и механизмов, заданных в штатном расписании.

Для того, чтобы найти такое решение, система может перемещать исходные ЗР во времени в заданных пределах. При этом должны соблюдаться следующие условия:

- длительность работ и потребность в ресурсах, указанные для каждой ЗР, должны сохраняться;
- порядок выполнения работ одного типа и для одного и того же объекта должен сохраняться;
- могут быть заданы ограничения по изменению сроков выполнения работ: например, нельзя изменять сроки более, чем на 7 дней в любом направлении, либо можно изменять не более, чем на 10 дней в сторону уменьшения, но не более, чем на 2 дня в сторону увеличения.

Результат:

В результате система должна предложить новые плановые даты выполнения каждой ЗР при условии соблюдения заданных ограничений. Если решения нет, то система должна сообщить об этом, желательно при этом

указать, какого ресурса или механизма (по мнению системы) недостаточно для выполнения плана.

1.5 Выводы

Были проанализированы два подхода к решению проблемы планирования в системах управления активами. Так как в самых популярных на сегодняшний день ЕАМ-системах, таких как 1С ERP, "ТРИМ", зарубежных SAP и Махіто есть лишь стандартные полуавтоматические способы планирования работ, то одним из самых востребованных направлений по улучшению такого функционала является полностью автоматическое планирование по заданным потребностям с учетом актуального распределения ресурсов.

Двумя самыми подходящими для такого функционала подходами являются мультиагентные системы и программирование в ограничениях. По результатам анализа подход с применением мультиагентных систем сложно рассматривать в качестве основного для данного типа задач, так как до сих пор, в том виде в каком он был представлен изначально, не показал реальных практических результатов, а также не было представлено каких-либо общедоступных сред для мультиагентного программирования. Программирование в ограничениях, напротив, предоставляет наиболее понятный набор инструментов и практик, поэтому для решения проблемы ресурсного планирования был выбран именно такой подход.

2 Проектирование

Проектирование планировщика ресурсов предполагает собой разработку, состоящую из следующих этапов:

- анализ доступных средств платформы .Net;
- выбор типа базы данных и её провайдера;
- определение алгоритмов, по которым будет работать программный модуль;
- написание и отладка программного модуля.

2.1 Проектирование архитектуры прототипа системы решения, включая разработку структуры (компоненты и связи между ними), внутреннего и внешнего функционирования.

2.1.1 Анализ доступных средств платформы .Net

На данный момент платформу .Net можно условно поделить на две части - .Net Framework и .Net Core.

На данный момент платформа .Net Framework является поддерживаемой, но устаревшей, и по рекомендациям Microsoft [17] не следует начинать новые проекты на этой платформе. Дополнительно необходимо указать, что платформа .Net Framework не является кроссплатформенной и работает только в операционных системах семейства Windows, что существенно ограничивает возможности современной разработки на ее основе на данный момент.

Открытый кроссплатформенный фреймворк .Net Core является преемником .Net Framework. Именно Core версию Microsoft рекомендует для всех новых проектов. Главным отличием новой платформы в первую очередь стала поддержка работы в Unix-системах (Linux, MacOS), соответственно все сервисы, написанные на .Net Core, без каких-либо ограничений могут быть

использованы в современной серверной парадигме: Linux серверы, Docker, контейнеризация и виртуализация.

Для данного проекта была выбрана самая актуальная версия фреймворка .Net Core - .NET 6, так как она является LTS-версией, а так же соответствует всем требованиям проекта - открытости и кроссплатформенности.

В качестве платформы для взаимодействия с пользователем был выбран серверный вариант фреймворка Blazor. Blazor - это бесплатный и открытый фреймворк для написания веб приложений на языке C#, поддерживающий реактивность, то есть работающий напрямую с DOM-элементами, а не HTML страницами. Blazor делится на Blazor WebAssembly - полностью клиентское приложение и на Blazor Server - клиент-серверную реализацию, в которой пользовательская и серверная часть общаются между собой по протоколу SignalR. SignalR - это бесплатная библиотека с открытым исходным кодом для Microsoft ASP.NET, которая позволяет серверному коду отправлять асинхронные уведомления веб-приложениям на стороне клиента. Библиотека включает в себя серверные и клиентские компоненты JavaScript.

2.1.2 Определение алгоритмов, по которым будет работать программный модуль

За планирование и/или перепланирование в планировщике будет отвечать библиотека Google OR-Tools. Задача, изложенная выше в первой главе, однозначно подходит под определение задач целочисленного программирования в ограничениях, так как каждый объект виртуальной модели в данной задаче можно определить как целочисленную переменную или их множество, на которые в явном виде можно наложить ряд ограничений и целевых функций, соответствующих условиям задачи.

Для такого типа задач пакет Google OR-Tools предоставляет отдельный класс для решения задач целочисленного программирования в ограничениях - CP-SAT, где CP (от англ. Constraint Programming - Программирование в ограничениях) - непосредственно целочисленное программирование в ограниче-

ниях, а SAT (от англ. Satisfiability - Удовлетворимость) - указывает, что целью решения данного класса задач является не нахождение одного конкретного результата, а нахождение одного или нескольких решений, удовлетворяющих ограничениям, наложенным на входные данные.

2.1.3 Выбор типа базы данных и её провайдера.

В первую очередь при выборе типа баз данных встаёт вопрос об использовании реляционных или нереляционных баз данных. В целом в ЕАМ-системах обычно используются именно реляционные базы данных, так как этому способствует необходимость в четкой иерархической структуре модели представления активов предприятия. Также в ЕАМ-системах редко встречается необходимость в шардинге - то есть в горизонтальном расширении данных для обеспечения масштабируемости. Исходя из этого для данного проекта была выбрана именно реляционная база данных.

В качестве реляционной СУБД был выбран PostgreSQL. PostgreSQL - это бесплатная система управления реляционными базами данных с открытым исходным кодом, в которой особое внимание уделяется расширяемости и соответствию стандарту SQL.

В данный момент ввиду острой необходимости в импортозамещении и независимости от зарубежных ПО, сервисов и средств именно СУБД PostgreSQL видится лучшим вариантом. У российских разработчиков уже есть большой опыт в развитии своих доработок и ответвлений PostgreSQL. Так, например, давно и успешно применяется ответвление Postgres Pro [18], СУБД ЛИНТЕР-ВС на базе PostgreSQL используется в ОС MCBC, а в дистрибутив операционной системы Astra Linux Special Edition (версия для автоматизированных систем в защищённом исполнении, обрабатывающих информацию со степенью секретности «совершенно секретно» включительно) включена СУБД PostgreSQL, доработанная по требованиям безопасности информации. Особенно важно, что СУБД Postgres PRO входит в реестр российского ПО [19].

2.1.4 Проектирование архитектуры

Желаемая архитектура системы представлена на рисунке 1. Предполагается, что разрабатываемая система ресурсного планирования будет являться отдельным сервером планирования, работающим независимо от ЕАМ-системы. Связь между ЕАМ-системой и сервером планирования должна осуществляться посредством наличия общей базы данных и API для передачи команд и промежуточных объектов.

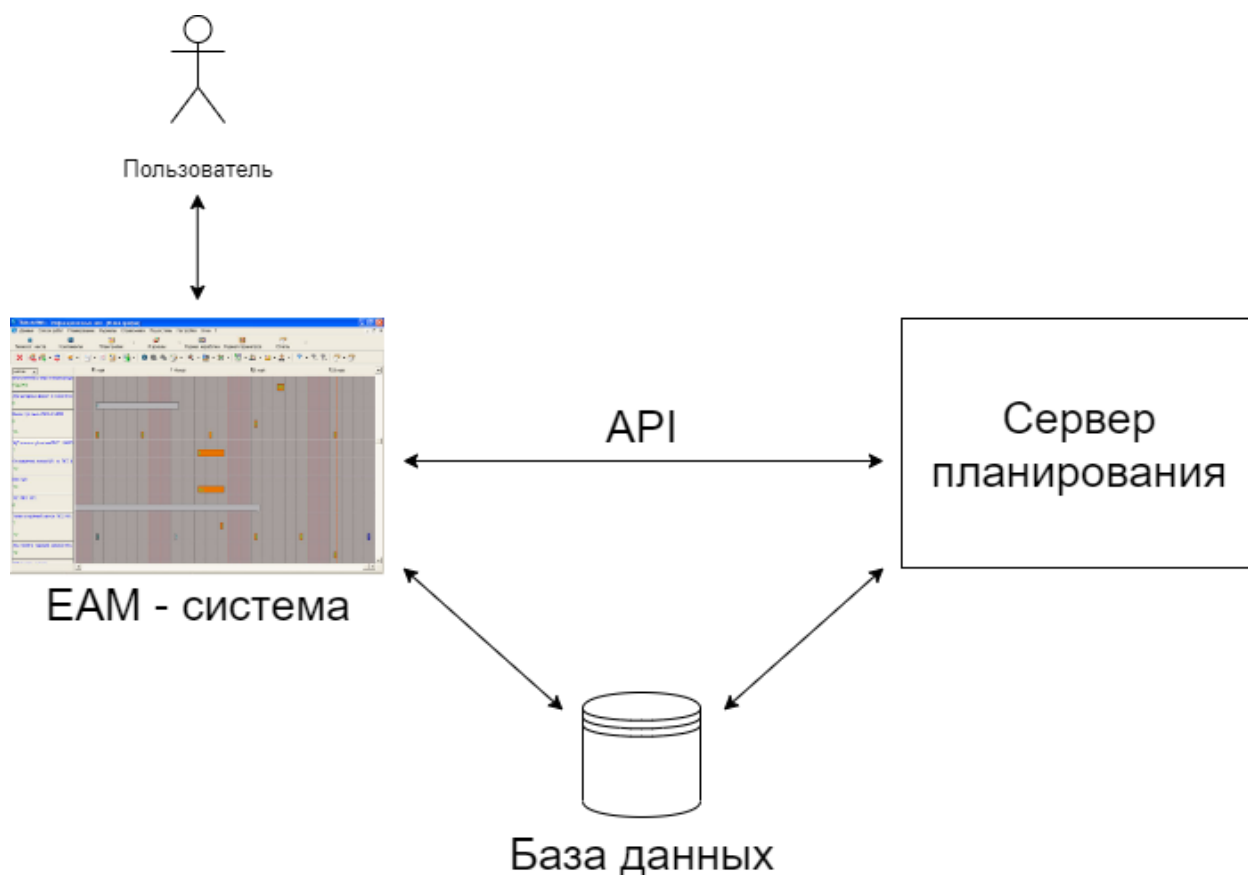


Рисунок 1 – Архитектура системы.

Архитектуру самого сервера планирования (рис. 2) можно представить как API, который посредством контроллера вызывает сервис перепланирования, использующий библиотеку OR-Tools.

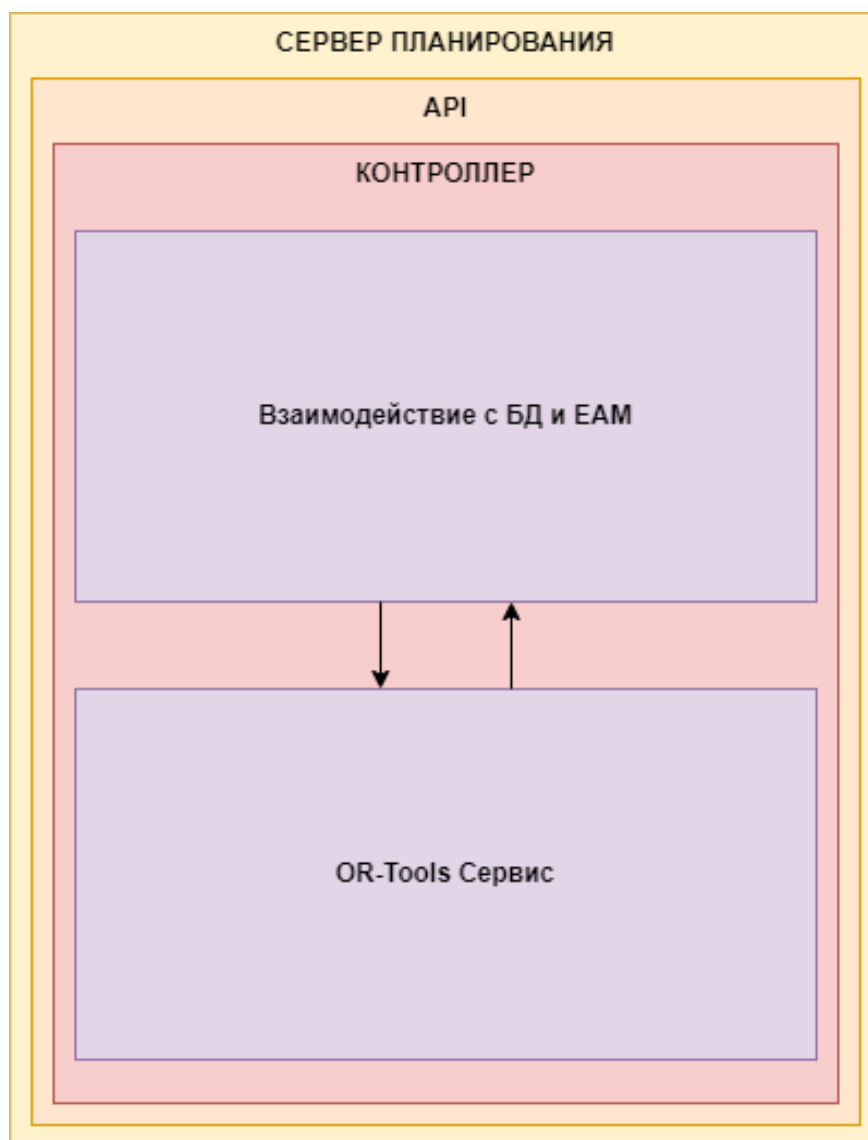


Рисунок 2 – Архитектура сервера планирования.

Рассмотрим представленную на рисунке 2 архитектуру сервера планирования более подробно.

Фронтенд (ЕАМ-система) отправляет команду через API, в котором в привязанном к маршруту контроллере выполняются определенные действия - инициализация перепланирования выбранных через фронтенд работ или их же, но уже перепланированных, отправка обратно во фронтенд (ЕАМ-систему) на отображение и подтверждение.

Фактически самой главной частью представленного сервера планирования является сервис OR-Tools. Именно данный сервис отвечает за основную часть - непосредственно процесс планирования предварительно задан-

ных работ. В данном сервисе происходит инициализация модели, задание ограничений для этой модели, а также задается целевая функция, то есть функция, относительно которой будут искаться все удовлетворяющие ограничениям решения для данной модели.

2.2 Разработка прототипов алгоритмов, структур данных.

2.2.1 Типы данных Google OR-Tools

Все объекты, для которых проводится процесс оптимизации, должны быть представлены в фреймворке Google OR-Tools одним из следующих типов данных:

- IntVar;
- BoolVar;
- IntervalVar;
- OptionalIntervalVar.

Тип данных IntVar это объект, который может принимать любое целочисленное значение в определенных диапазонах, такой диапазон называется доменом. В коде объявляется как

```
var integer_variable = model.NewIntVar(0, 100,  
    ↪ "Integer Variable");
```

где 0 и 100 это домен переменной, а "Integer Variable" - произвольное имя переменной, задаваемое пользователем.

Тип данных BoolVar представляет собой переменную для хранения логических значений. Внутри фреймворка BoolVar аналогичен типу IntVar, но с доменом $[0, 1]$. Выделен в отдельный тип данных для удобства программирования.

Пример объявления:

```
var boolean_variable1 = model.NewBoolVar("Boolean  
    ↪ Variable");
```

В качестве параметров при объявлении передается только произвольное имя переменной, так как домен по умолчанию лежит в пределах $[0, 1]$

Тип `IntervalVar` представляет собой интервальную переменную. Интервальная переменная - это и ограничение, и переменная. Она определяется тремя целочисленными переменными: `start`, `size` и `end`.

Она является ограничением, потому что внутри она обеспечивает выполнение того, что `start + size == end`.

Это также переменная, поскольку она может задаваться в определенных методах ограничения планирования, таких как: `NoOverlap`, `NoOverlap2D`, `Cumulative`.

Объявляется так:

```
var interval_variable = model.NewIntervalVar(start,  
→ size, end, "Interval Variable");
```

в которой `start` - предполагаемый момент начала интервала, `size` - размер (длина) интервала - обычно фиксированное целое число, `end` - предполагаемый момент окончания интервала.

В качестве значений `start` и `end` могут приниматься либо целые числа - в таком случае, например, суть оптимизации такой переменной заключается в нахождении её места на горизонте планирования, либо переменные типа `IntVar` - тогда оптимизация данной переменной будет зависеть и от также неизвестных переменных определенного домена.

Последним из доступных типов является `OptionalIntervalVar` - опциональный интервал. Этот отличается от `IntervalVar` литералом `is_present`, который указывает, активна (то есть фактически присутствует в модели и участвует в процессе оптимизации) данная переменная или нет.

Пример:

```
var optional_interval_variable =  
→ model.NewOptionalIntervalVar(start, size, end,  
→ is_present "Optional Interval Variable");
```


в которой `is_present` представляет собой логическая переменную типа `BoolVar`.

Представленные выше типы данных определяются как часть модели. Модель это некторое условное пространство имён, которое содержит в себе переменные-объекты модели, методы для работы с объектами, ограничения для них, а также методы для задания целевой функции модели.

Решение модели эквивалентно нахождению для каждой переменной единственного значения из множества начальных значений (называемого начальной областью), такого, что модель выполнима, или оптимальна, если вы предоставили целевую функцию.

Основными ограничениями для модели можно выделить:

- `Add`;
- `AddNoOverlap`;
- `AddAllDifferent`;
- `AddImplication`.

Метод `Add` добавляет ограничение в виде линейного выражения вида $x + y = z$, где x, y, z - целые числа или переменные типа `IntVar`.

Ограничение `AddNoOverlap` гарантирует, что все выбранные для него интервалы не пересекаются во времени.

`AddAllDifferent` заставляет все переменные иметь разные значения.

Метод `AddImplication` обеспечивает ограничение вида ЕСЛИ, ТО - $a \Rightarrow b$.

Класс `LinearExpr` позволяет производить над типами данных `OR-Tools` различные математические операции, такие как сумма, скалярное произведение, умножение на коэффициент - `Sum()`, `ScaleProd()`, `Term()`.

Для задания целевой функции фреймворк предлагает два метода - `maximize` и `minimize`. Данные методы позволяют максимизировать или минимизировать некоторые необходимые параметры модели.

2.2.2 Определение требований к модели

Так как при планировании работ периоды занятости ресурсов могут накладываться друг на друга независимо от качества планирования, то ожидается, что не все работы смогут быть запланированы. Соответственно, цель работы планировщика заключается в максимизации количества запланированных работ и выражается формулой:

$$\max \sum_0^N (n_0, \dots, n_N) \quad (2.1)$$

где N — количество работ;

n — переменная домена $[0, 1]$, означающая присутствие работы.

Так как работы одного типа и работы, привязанные к одному человеку, никак не должны пересекаться, то должно так же выполняться следующее ограничение:

$$\forall p \in I \quad \forall q \in I : p \neq q \implies V_p \cap V_q \neq \emptyset \quad (2.2)$$

где V_i — семейство интервалов;

p, q — интервал множества V .

Из дополнительных ограничений следует отметить:

Для каждого рабочего количество запланированных работ должно быть меньше или равно изначальному количеству.

Так как в типах работ присутствуют плановые работы, то для работ таких типов должна соблюдаться определенная периодичность, например, планирование раз в день, неделю, месяц и так далее.

3 Реализация предлагаемого решения и его оценка

Реализация системы ресурсного планирования состоит в описании модели методами целочисленного программирования, интеграции этой модели в сервис планирования и написания подобия ЕАМ-фронтенда, который будет взаимодействовать с сервисом, то есть отображать исходные данные, запускать процесс перепланирования, отображать результаты работы сервиса, а также различную статистику.

3.1 Реализация прототипа технического решения

3.1.1 Описание инструментов и форматов данных

В качестве среды программирования была выбрана Visual Studio 2022 под операционной системой Windows 11.

Библиотека Google Or-Tools поставляется из официального NuGet репозитория.

Как сервис ресурсного планирования, так и частичный функционал подобия ЕАМ-системы будут реализованы с помощью Blazor Server.

Данные о ресурсах и работах будут храниться в реляционной базе данных PostgreSQL, взаимодействие с базой данных извне будет происходить через SQL клиент DBeaver, а внутри проекта через ORM Entity Framework Core.

Структурная схема использованных инструментов представлена на рисунке 3.

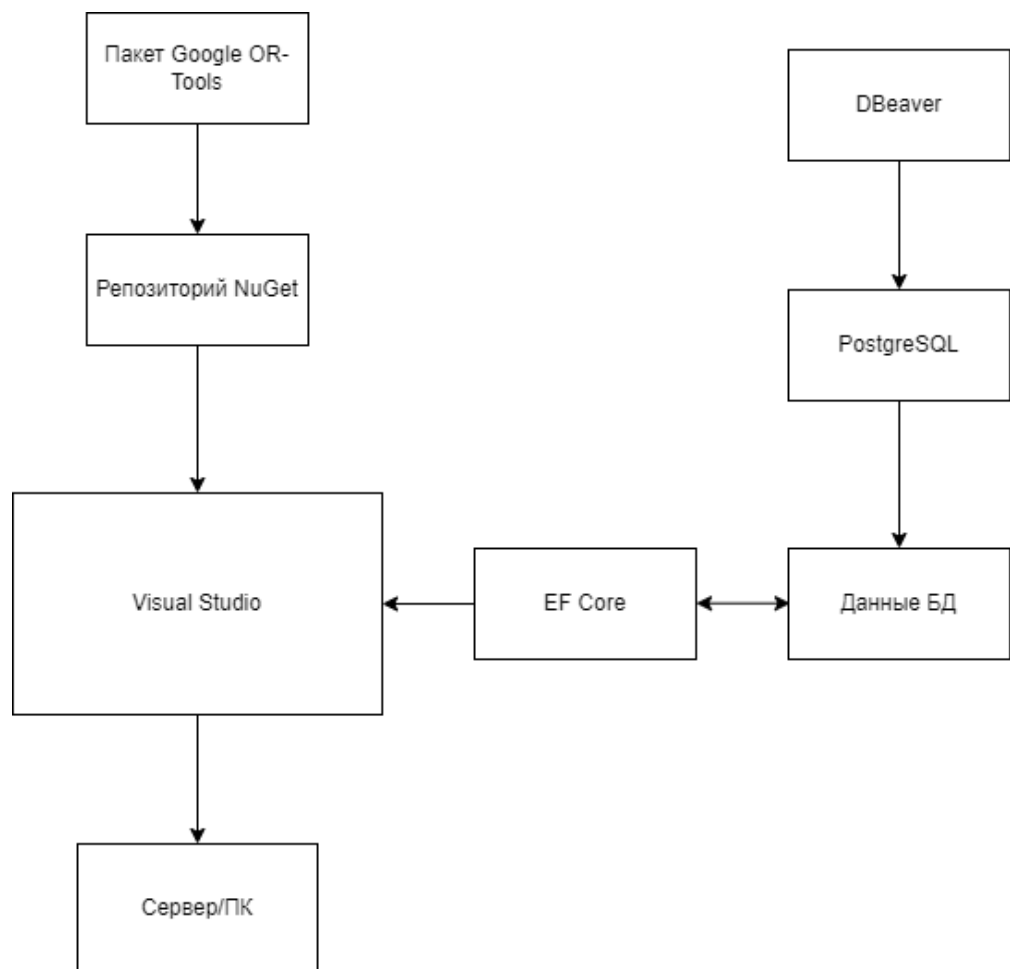


Рисунок 3 – Структура взаимосвязей инструментов.

3.1.2 Описание модели базы данных

Схема реляционной базы данных, достаточной для описания модели, представленной в главе 1 представлена на рисунке 4.

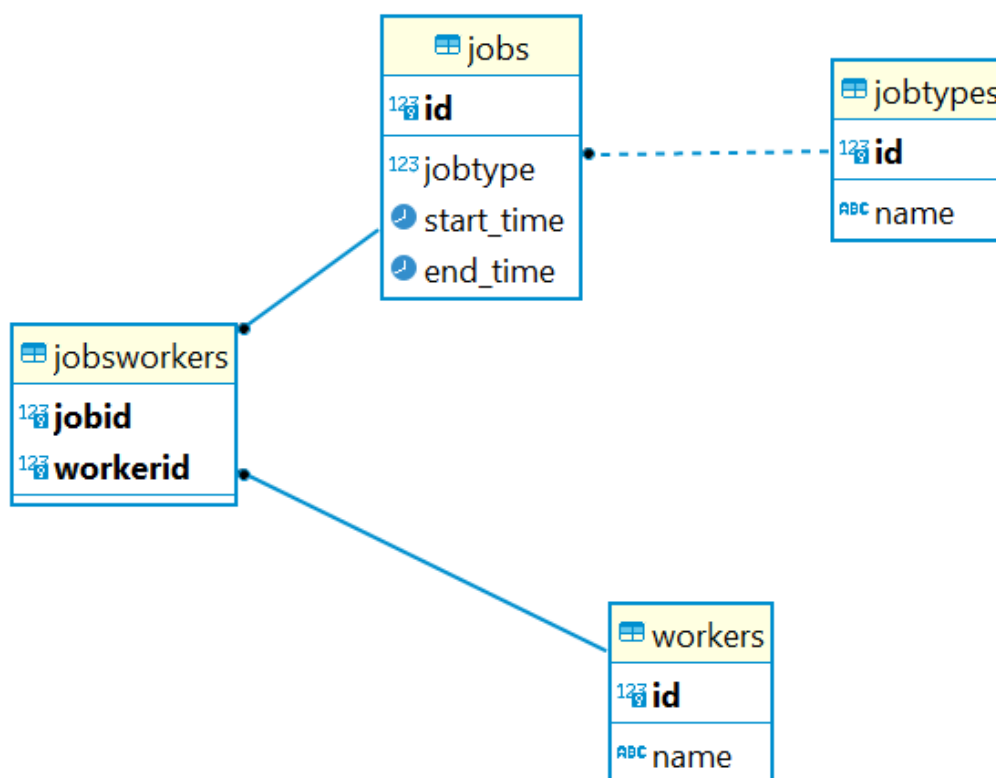


Рисунок 4 – Схема базы данных.

В схеме представлены следующие сущности:

- jobs
- jobtypes
- workers
- jobsworkers

Сущность jobs представляет все работы, запланированные на текущий момент времени.

Сущность jobtypes представляет все доступные типы работ. Связана с сущностью jobs связью один ко многим.

В таблице workers находятся все рабочие, которые могут быть привязаны к работам. Связана с таблицей jobs связью многие ко многим, поэтому необходимо использование промежуточной таблицы jobsworkers.

3.1.3 Прототипирование ЕАМ-системы

Для упрощения взаимодействия с сервисом планирования, а так же для упрощения написания вспомогательных инструментов, в качестве платформы для ЕАМ-системы был выбран веб-фреймворк Blazor Server.

При использовании модели хостинга Blazor Server приложение выполняется на сервере из приложения ASP.NET Core. Обновления пользовательского интерфейса, обработка событий и вызовы JavaScript обрабатываются через соединение SignalR.

Данный фреймворк позволяет создавать реактивные веб-приложения без использования языка JavaScript и его фреймворков, что значительно упрощает и ускоряет разработку веб-приложений для .Net программистов, не владеющих стеком технологий, таких как JavaScript, Angular, React и подобных.

Желаемый вид ЕАМ-системы представлен на рисунке 5:

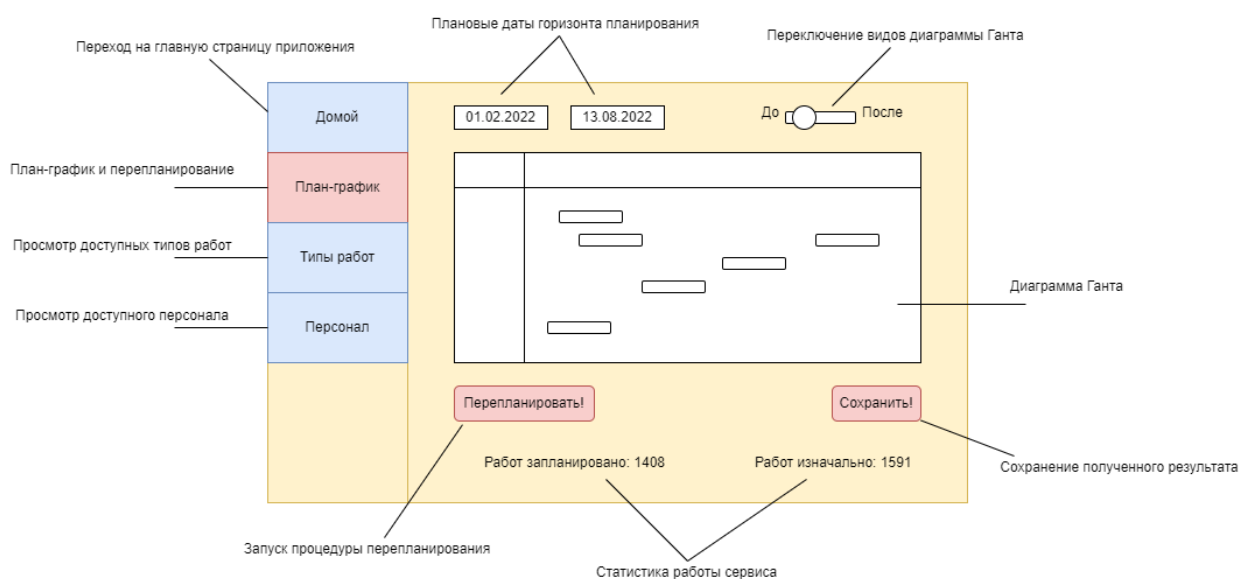


Рисунок 5 – Желаемый интерфейс прототипа ЕАМ-системы

На рисунке показано основное окно системы - окно планирования работ, в котором находятся элементы управления для выбора плановых дат, перепланирования и записи результата в базу данных. Основным информационным элементом на данной странице является диаграмма Ганта.

Страница "Домой" служит для отображения основной информации о прототипе системы, страницы "Типы работ" и "Персонал" служат только для отображения исходных данных по сущностям "Типы работ" и "Персонал".

Диаграмма Ганта - столбчатая диаграмма, которая используется для иллюстрации плана или графика работ, например, во времени. Данный тип диаграмм повсеместно используется в системах подобного класса и очень удобен для отображения информации данного типа.

Так как диаграмма Ганта достаточно сложный к реализации элемент, было принято решение использовать его готовую реализацию. Готовая реализация была взята из пакета Syncfusion [20], который бесплатно предоставляет ряд веб-компонент для некоммерческого использования и для студентов. Вид диаграммы из пакета Syncfusion Blazor представлен на рисунке 6.

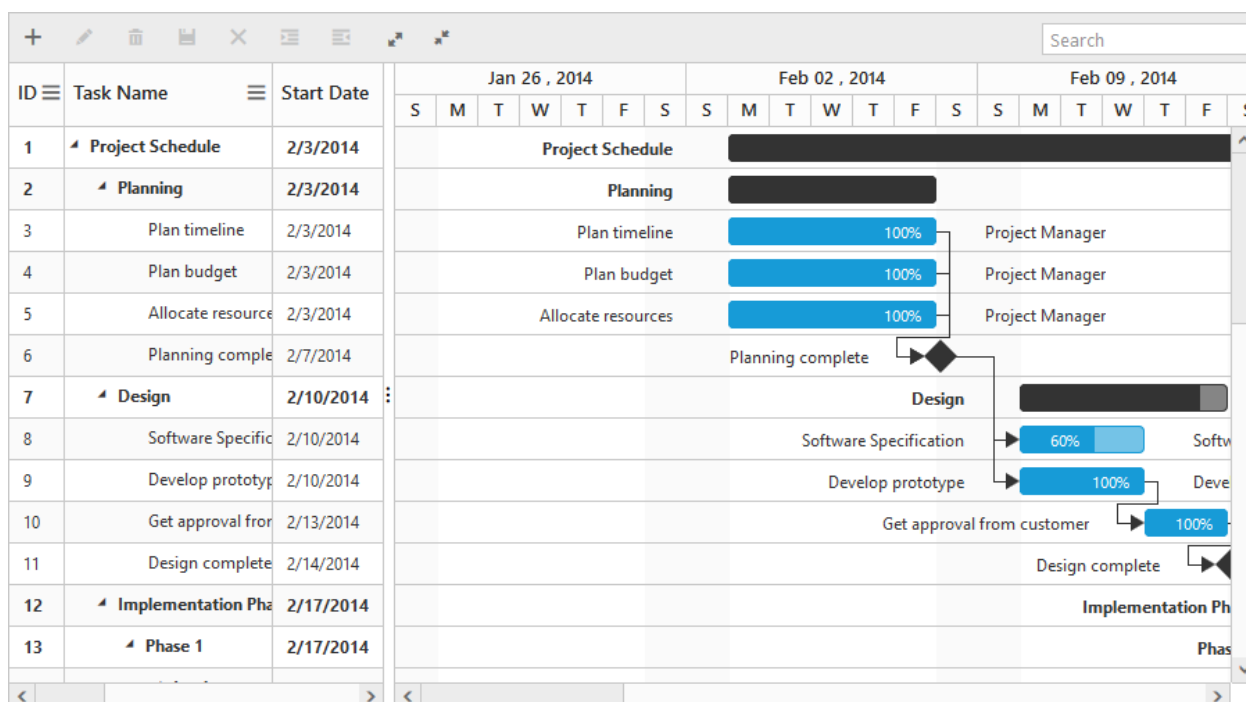


Рисунок 6 – Компонент диаграммы Ганта из пакета Syncfusion

Итоговый вид основного окна представлен на рисунке 7.

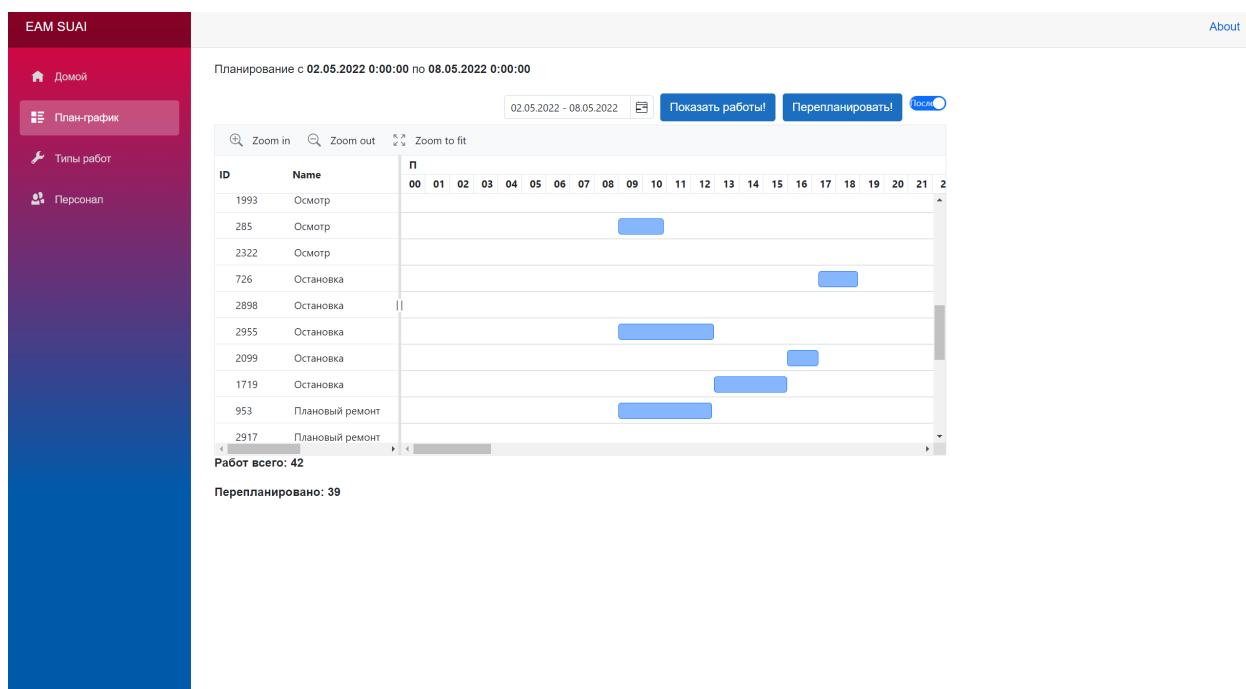


Рисунок 7 – Страница планирования работ прототипа ЕАМ-системы

От кнопки сохранения перепланированных работ было решено отказаться, так как данный функционал является несущественным в рамках данной работы.

Остальные страницы прототипа представлены на рисунках 8, 9 и 10.

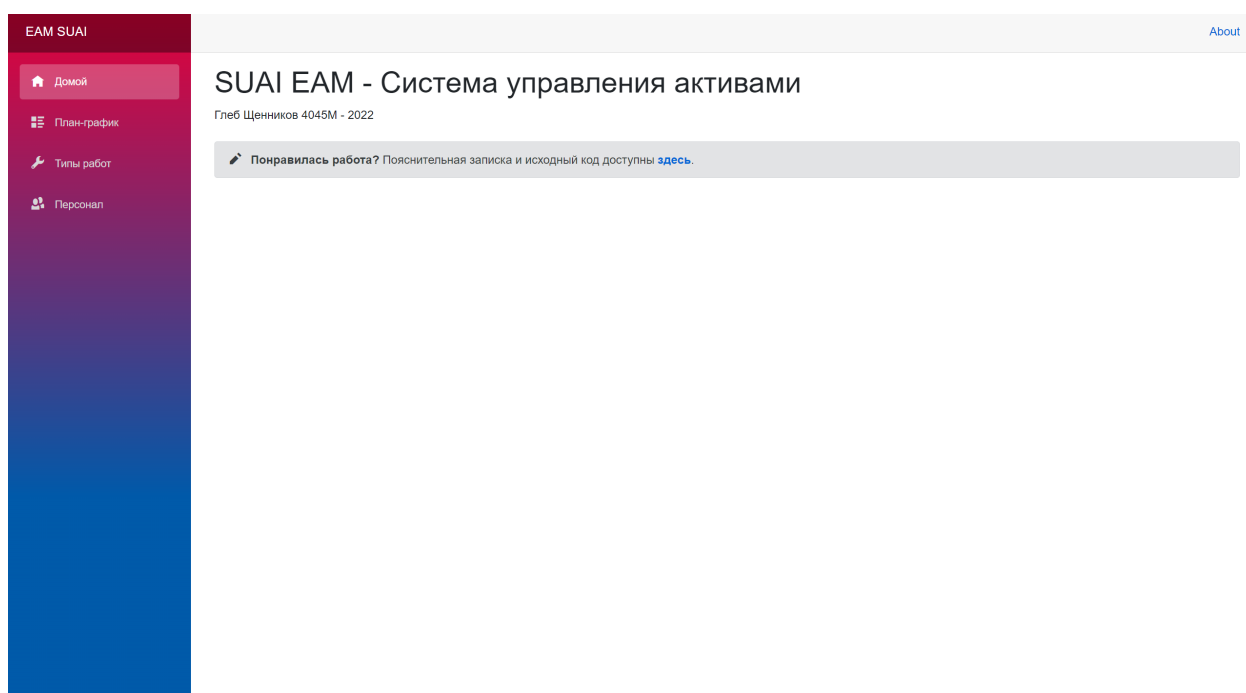


Рисунок 8 – Домашняя страница прототипа

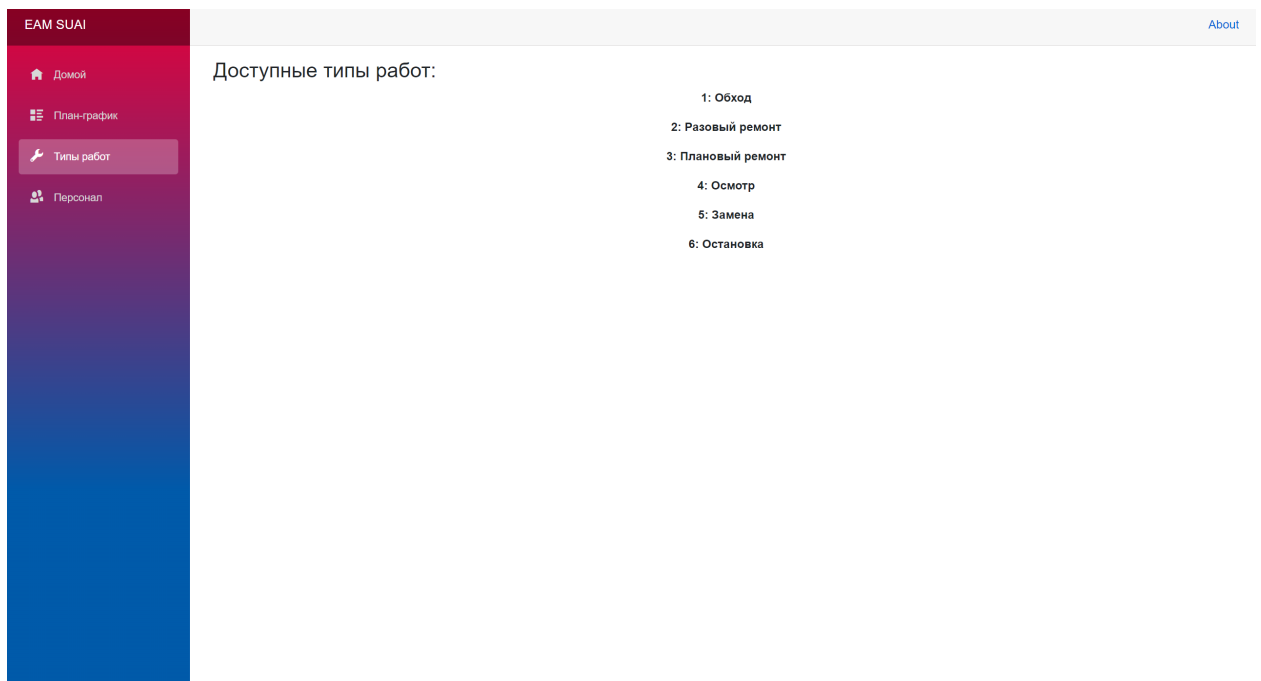


Рисунок 9 – Страница ”Типы работ”

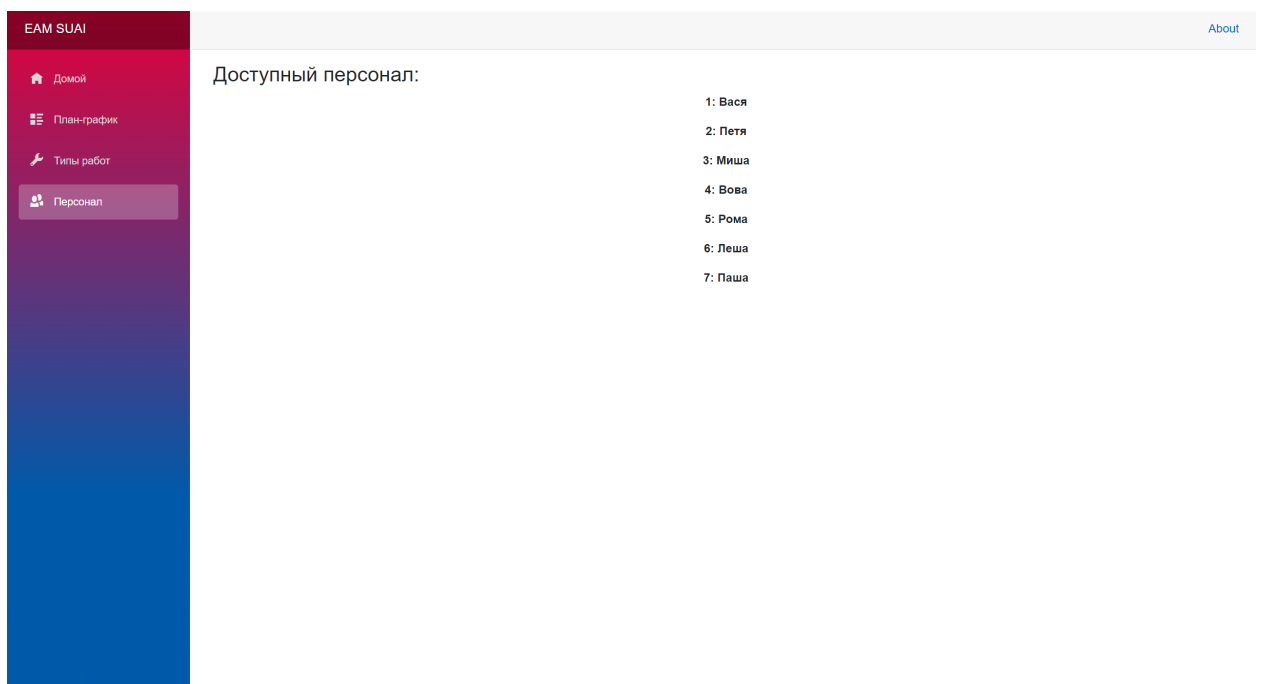


Рисунок 10 – Страница ”Персонал”

Главным преимуществом использования фреймворка Blazor в подобных проектах является отсутствие необходимости в знании каких-либо дополнительных языков и их средств, например, JavaScript и React - вся реализация веб-приложения состоит целиком из HTML/CSS и кода на языке C#.

В качестве примера можем рассмотреть страницу планирования работ. Данная страница состоит из двух частей: Razor страницы/компонента с расширением .razor или .cshtml и вспомогательного кода, наподобие модели-представления из паттерна MVVM. Вспомогательный код может находиться либо прямо внутри представления, либо отдельным файлом как составной класс данного представления. Такая структура представлена на рисунке 11.

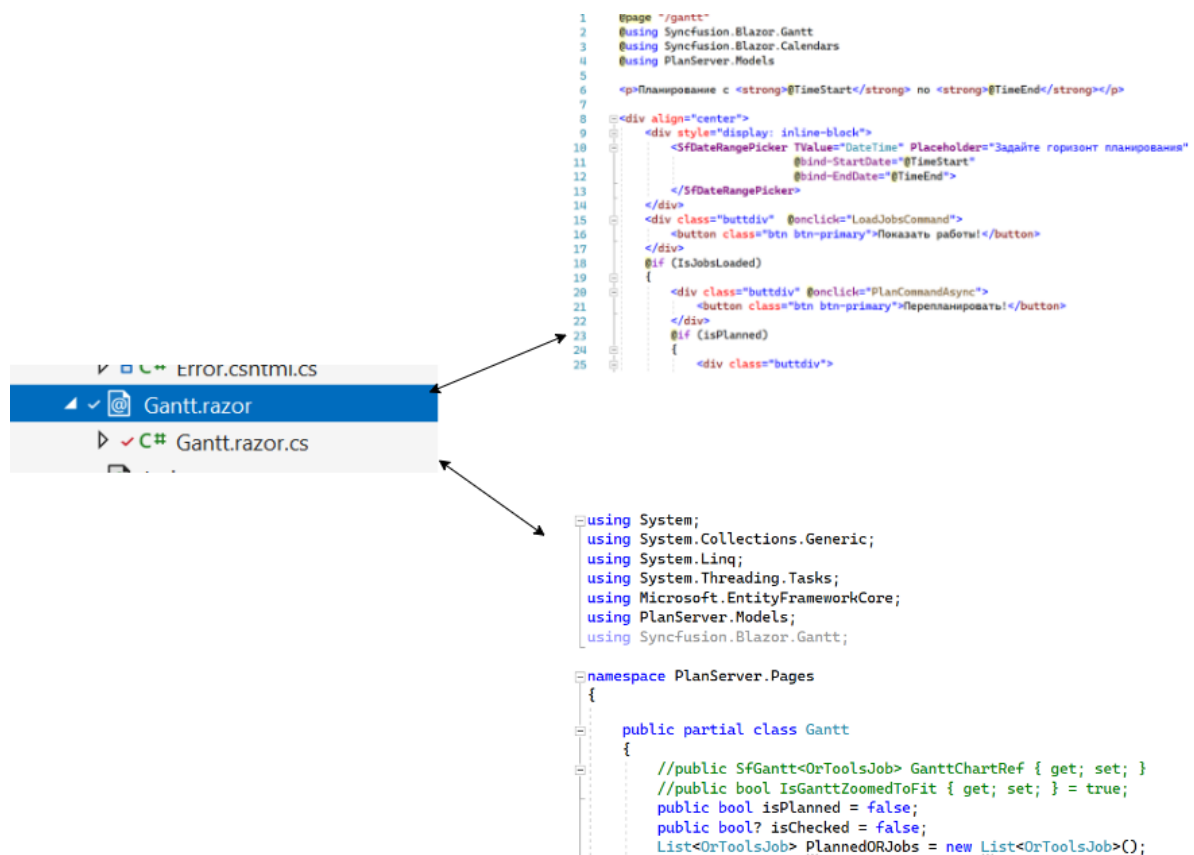


Рисунок 11 – Страница "Персонал"

Как видно из рисунка выше, в представлении Gantt.razor задан HTML-код со вставками на C#. Данные вставки берутся из методов и свойств составного класса Gantt.razor.cs и позволяют реактивно менять значения на странице.

3.1.4 Моделирование системы средствами программирования в ограничениях

Все действия по планированию и/или перепланированию работ происходят на горизонте планирования, выбираемого пользователем, и который будет определяться целочисленным типом данных. Горизонт планирования рассчитывается в минутах и находится в диапазоне от 0 до 2,147,483,647 (ограничивается целочисленным типом данных языка C#). Был выбран горизонт планирования в минутах, так как он обеспечивает достаточную точность относительно часов, а в численном виде значительно меньше секунд, что сократит необходимую память для планирования.

Для описания работ были выбраны следующие типы данных, доступные из библиотеки OR-Tools:

- BoolVar
- IntVar
- OptionalIntervalVar

Тип данных IntVar в данной модели определяет переменные моментов начала и конца работ. Лежит в диапазоне от 0 до значения горизонта планирования.

Типом BoolVar определяется актуальность работы, то есть ее участие в процессе планирования. Данная переменная используется в типе OptionalIntervalVar. Переменная задается в диапазоне от 0 до 1 по умолчанию, соответственно если значение переменной равно единице, то работа учитывается при планировании, 0 - не учитывается.

OptionalIntervalVar уже определяет саму работу на горизонте планирования. В отличие от просто IntervalVar данный тип - опциональный и зависит от переменных типа BoolVar, используемых литералом `is_present`.

Для составления модели до момента задания ограничений необходимо в первую очередь инициализировать работы, то есть инициализировать все переменные, представляющие работы.

В первоначальной версии модели инициализация была проделана так:

```
for h in range(horizon)
    ...
```

```

for w in workers
    ...
    for jt in job_types
        ...
        for job in jobs

```

В данном варианте инициализация происходит относительно четырех сущностей модели: каждой минуты горизонта планирования, каждого рабочего, каждого типа работы и непосредственно каждой работы.

Отсюда, например, при 10 рабочих, 10 типах работ, 10 запланированных работах и горизонте планирования в 10080 минут (семь дней):

$$10 * 10 * 10 * 10080 = 10080000$$

Получаем некий объект, в котором уже для простейшего примера ресурсного планирования получим непомерно огромные затраты по памяти, причем полученную цифру в 10080000 необходимо еще умножить на 4, так как имеем при инициализации для каждой работы две переменных типа IntVar, одну BoolVar и одну OptionalIntervalVar, в итоге - свыше сорока миллионов объектов для простого тестового примера.

Необходимость в наличии инициализации относительно горизонта планирования была связана с недопониманием концепции использования опциональных интервалов. Изначально казалось необходимым наличие возможности начала работы в каждую из минут горизонта планирования, что соответственно добавляло лишнее измерение в инициализацию.

Тем не менее, такой подход оказался рабочим и выдавал удовлетворяющий результат, то есть соответствовал ограничениям и целевым функциям модели, откуда можем сделать вывод о возможности наличия более чем одного подхода в моделировании при решении проблем программирования в ограничениях.

Результаты исследований прототипа модели первой версии будут представлены ниже в соответствующей главе.

Во второй итерации модели, уже при переносе с языка Python на C#, были найдены некоторые недочеты первой версии модели, а также пересмотрена процедура инициализации, в результате которой был убран цикл по горизонту планирования. Это обуславливается тем, что в интервальных переменных и так задается их момент возможного начала и момент возможного конца, соответственно нет необходимости в привязке начала интервалов к конкретному моменту горизонта планирования.

Итоговый вид инициализации на языке C#:

```
...
foreach (var w in workers)
{
    ...
    foreach (var jt in job_types)
    {
        ...
        foreach (var job in jobs)
        {
            if (jt == job.Type && w == job.WorkerID)
            {
                var performed_at =
→ model.NewBoolVar($"performed by {w}wID {jt}type
→ {job.ID}jID");
                var start = model.NewIntVar(0, horizon,
→ $"start of {w} {jt} {job.ID}");
                var end = model.NewIntVar(0, horizon,
→ $"end of {w} {jt} {job.ID}");
                var optint =
→ model.NewOptionalIntervalVar(start,
→ job.TimeSpanMinutes, end, performed_at, $"interval
→ of {w} {jt} {job.ID}");
```

```

        ...
    }
}

```

Следующим шагом после инициализации входных данных для модели является задание ограничений.

Первая группа ограничений - работы одного типа, а так же работы одного рабочего не должны пересекаться.

Для этого языком подзапросов LINQ из переменных, представляющих входные данные необходимо достать необходимые работы. Для рабочих:

```

//работы по рабочему
List<List<IntervalVar>> jobIntervals_by_worker =
    ↪ new();
foreach (var w in intervals_by_worker)
{
    var flatlist = w.SelectMany(i => i).ToList();
    jobIntervals_by_worker.Add(flatlist);
}

//работы одного рабочего не пересекаются
foreach (var ji in jobIntervals_by_worker)
{
    ji.AddRange(sleepIntervals);
    model.AddNoOverlap(ji);
}

```

В данном примере следует обратить внимание на метод AddNoOverlap, который и устанавливает ограничение на непересечение элементов выборки ji, то есть множества работ одного рабочего.

Задание ограничений этой группы для работ одного типа происходит аналогично.

Дополнительно следует отметить, что для множества рабочих было задано дополнительное ограничение в виде работы только по будням с девяти утра до девяти вечера.

```
//определим нерабочие (ночные интервалы) - допустим с
→ 9 утра до 9 вечера
//даты планирования выбираются пользователем с 00:00
→ по 00:00
//0 + 540 => ночь; +720 => день; + 180 => остатки ночи
List<IntervalVar> sleepIntervals = new();
int dayCounter = 0;

foreach (DateTime day in EachDay(reftime, lastday))
{
    int st = dayCounter * 1440;
    if ((day.DayOfWeek == DayOfWeek.Saturday) ||
→ (day.DayOfWeek == DayOfWeek.Sunday))
    {
        //по выходным не работаем
        int fin1 = st + 1440;
        var siSS = model.NewIntervalVar(st, 1440, fin1,
→ $"sat/sun {dayCounter}");
        sleepIntervals.Add(siSS);
    }
    else
    {
        //по будням - с 9 до 9
        int fin = st + 540;
        var si = model.NewIntervalVar(st, 540, fin,
→ $"sleep 1 of day {dayCounter}");
        sleepIntervals.Add(si);
    }
}
```

```

        st = fin + 720; //пропускаем рабочие 12 часов
        fin = st + 180;
        //второй ночной интервал
        var si2 = model.NewIntervalVar(st, 180, fin,
→ $"sleep 2 of day {dayCounter}");
        sleepIntervals.Add(si2);
    }
    dayCounter++;
}

```

Следующим шагом необходимо ограничить возможное количество работ. В данном варианте мы считаем, что перепланированное количество работ одного типа и количество работ у одного рабочего должно быть меньше или равно исходному их количеству. Тогда для работ одного типа:

```

///разделяем performed_at по типам работы
Console.WriteLine("разделяем performed_at по типам
→ работы");
List<List<IntVar>> performedAt_by_type = new();
foreach (var t in job_types)
{
    List<IntVar> perflist = new();
    performedAt_by_type.Add(perflist);
}
foreach (var b_worker in bools_by_worker)
{
    for (int b_type = 0; b_type < b_worker.Count;
→ b_type++)
    {
        var flatperf = b_worker[b_type];
        performedAt_by_type[b_type].AddRange(flatperf);
    }
}

```



```

}
for (int i = 0; i < performedAt_by_type.Count; i++)
{
    model.Add(LinearExpr.Sum(performedAt_by_type[i])
    ↪  <= job_type_num[i]);
}

```

Необходимое ограничение задается в строке `model.Add(LinearExpr.Sum(performedAt_by_type[i]) <= job_type_num[i])`, где метод `Add()` задает для модели некое линейное ограничение. Метод `LinearExpr.Sum()`, переданный в `Add()` позволяет производить над множествами объектов типа OR-Tools операцию сложения значений элементов множества. Соответственно данной строкой мы задаем для модели, что количество работ перепланированных меньше или равно количеству работ исходных.

Ограничение по количеству работ для конкретных рабочих задается аналогично.

Так как у работ может быть задана некая периодичность выполнения, например, каждый день, раз в неделю, раз в месяц и так далее, то необходимо задать механизм эту периодичность устанавливающий.

Изначально данное ограничение представлялось очень простым. У каждой работы (опционального интервала) заданы переменные `start` и `end` типа `IntVar`, обозначающие возможный момент начала и возможный момент окончания работы. Соответственно ограничение выглядело тривиально: $start_{n+1} - end_n > X$, где X - интервал, через который плановые работы одного типа должны выполняться.

Так как переменные `start` и `end` используются в опциональном интервале, то ожидалось, что при невозможности для двух работ соответствовать ограничению $start_{n+1} - end_n > X$ - данные работы просто не будут учитываться, так как привязаны к типу `OptionalIntervalVar`. В действительности же заданные ограничения при невозможности их удовлетворить делали всю

модель невыполнимой, в связи с тем, что ограничения к данным переменным рассматриваются сами по себе и без привязки к другим типам в модели. Именно для таких случаев используются связывающие ограничения.

Связывающие ограничения (от англ. - channeling constraint) связывают переменные внутри модели. Они используются, когда необходимо выразить сложную связь между переменными, например, "если эта переменная удовлетворяет условию, то она заставляет другую переменную принять определенное значение".

В данном случае необходимо указать, что если ограничение $start_{n+1} - end_n > X$ выполнимо, то литерал `is_present`, отвечающий за актуальность опционального интервала, станет равным единице, то есть работа с этим литералом будет учитываться при планировании.

Выражается таким образом:

```
//для третьего типа (плановая работа) - работы не чаще
→ раза в день
...
if(type == 3)
{
    for (int i = 0; i < startAt_by_type[k].Count -
→ 1; i++)
    {
        for (int j = i+1; j <
→ startAt_by_type[k].Count; j++)
        {
            //if (b-a>1440) => b=0
            model.Add(startAt_by_type[k][j] -
→ endAt_by_type[k][i] >=
→ 1440).OnlyEnforceIf(performedAt_by_type[k][i]);
        }
    }
}
```

```
        break;
    }
    ...

```

В этом примере с помощью метода `OnlyEnforceIf()` устанавливается, что если выполнятся ограничение в методе `Add()`, то переменная типа `BoolVar` внутри метода `OnlyEnforceIf()` обратится в 1.

Последним шагом остается задать целевую функцию. В качестве целевой функции предлагается максимизировать количество перепланированных работ в целом. Для этого напишем такой код:

```
//возьмем множество всех работ
var flatbools = bools_by_worker.SelectMany(i =>
    i).SelectMany(i => i);
//целевая функция - максимизировать количество работ
model.Maximize(LinearExpr.Sum(flatbools));

```

В данном примере целевая функция задается через метод модели `Maximize()`, в который передается множество всех работ. В данном случае каждый объект элемента множества является типом данных `BoolVar`.

3.2 Экспериментальные исследования прототипа решения

После создания прототипа была проведена экспериментальная оценка пригодности системы для задач ресурсного планирования в соответствии с поставленной задачей.

Основные характеристики аппаратно-программной конфигурации, которая использовалась для создания прототипа: компьютер – AMD Ryzen 9 5900HX, видеокарта MSI GeForce RTX 3070 Laptop 8Гб, оперативная память 16Гб DDR4.

Основными критериями для исследований предлагается использовать:

- Зависимость времени планирования от количества работ/ресурсов
- Зависимость требуемой памяти от количества работ/ресурсов

- Результативность алгоритма (процент количества перепланированных работ от исходного)

Для начала рассмотрим время планирования (рисунок 12) и требуемый объем памяти (рисунок 13) для планирования 13 работ на горизонте в одну неделю для моделей первой и второй версии:



Рисунок 12 – Разница во времени планирования для моделей первой и второй версии (меньше - лучше)



Рисунок 13 – Разница в требуемой оперативной памяти для моделей первой и второй версии (меньше - лучше)

Рассказываю почему так

Рассказываю про тесты нормальной модели

ЗАКЛЮЧЕНИЕ

Тоже что-то умное

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ (СТАРЫЙ)

1. Видеозапись Intel Fab Tour, [Электронный ресурс], URL: <https://www.youtube.com/watch?v=2ehSCWoaOqQ> (дата обращения 13.04.2022).
2. Virginia Dignum, Frank Dignum, "Agents are Dead. Long live Agents!", UmeåUniversity International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC AAMAS 2020, May 9–13, Auckland, New Zealand
3. Frank Dignum, Jeff Bradshaw, Barry Silverman, Willem van Doesburg, "Agents for Games and Simulations", Изд-во Springer, 2009. – 247с.
4. Городецкий В.И., Скобелев П.О, "Многоагентные технологии для промышленных приложений: реальность и перспектива", Труды СПИИ-РАН. 2017. № 55 (6). С. 11-45.
5. Сайт Wikipedia - Исследование операций [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Исследование_операций (дата обращения 23.02.2022).
6. Сайт фреймворк GLPK [Электронный ресурс], URL: <https://www.gnu.org/software/glpk/> (дата обращения 13.03.2022).
7. Сайт LP_Solve [Электронный ресурс], URL: <http://lpsolve.sourceforge.net/> (дата обращения 13.03.2022).
8. Сайт minizinc [Электронный ресурс], URL: <https://www.minizinc.org/> (дата обращения 13.03.2022).
9. Сайт OR-Tools [Электронный ресурс], URL: <https://developers.google.com/optimization> (дата обращения 13.03.2022).
10. Сайт CHOCO [Электронный ресурс], URL: <https://choco-solver.org/> (дата обращения 13.03.2022).

11. Сайт IBM CPLEX [Электронный ресурс], URL: <https://www.ibm.com/analytics/cplex-optimizer> (дата обращения 13.03.2022).
12. Сайт GUROBI [Электронный ресурс], URL: <https://www.gurobi.com/> (дата обращения 13.03.2022).
13. Giacomo Da Col, Erich C. Teppan, "Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem", Electronic Proceedings in Theoretical Computer Science. 2019. С. 259-265.
14. Hvattum, Lars Magnus & Løkketangen, Arne & Glover, Fred, Comparisons of Commercial MIP Solvers and an Adaptive Memory (TabuSearch) Procedure for a Class of 0–1 Integer Programming Problems, Algorithmic Operations Research Vol.7 (2012) 13–20.
15. Сайт Minizinc challenge [Электронный ресурс], URL: <https://www.minizinc.org/challenge.html> (дата обращения 11.04.2022).
16. Википедия - Диета Стиглера [Электронный ресурс], URL: https://en.wikipedia.org/wiki/Stigler_diet (дата обращения 03.04.2022).
17. .NET vs. .NET Framework for server apps [Электронный ресурс], URL: <https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server> (дата обращения 08.04.2022).
18. Сайт Postgres PRO [Электронный ресурс], <https://www.postgrespro.ru/> (дата обращения 08.04.2022).
19. Реестр - СУБД Postgres PRO [Электронный ресурс], https://reestr.digital.gov.ru/reestr/301574/?sphrase_id=1426840 (дата обращения 08.04.2022).
20. <https://www.syncfusion.com/blazor-components> [Электронный ресурс], https://reestr.digital.gov.ru/reestr/301574/?sphrase_id=1426840 (дата обращения 23.02.2022).