Содержание

Введение

Актуальность темы исследования. В рамках Четвертой промышленной революции или по-другому – Индустрии 4.0 в мире происходит постепенное внедрение киберфизических систем в жизнь человека. Данная тенденция ведет к растущей потребности в данных системах и их состовляющих: физической (датчики) и программной (вычислительные системы). Это обуславливает создание интеллектуальных вычислительных систем, которые смогут обеспечить большую точность и скорость планирования деятельности производств, что позволит сократить потери производственных мощностей и ресурсов, а также ускорит актуализацию планов по результатам деятельности предприятия. Из всего вышесказанного следует необходимость в разработке компонент для вышеописанных интеллектуальных систем, которые будут решать поставленные задачи, такие как моделирование производственных ресурсов, преобразование выходных данных, хранение данных и другие.

Цель работы. Целью данной работы является разработка и тестирование компонент системы планирования производства. Были поставлены следующие задачи:

— разработка модели ресурса сборочной линии;

— разработка модуля отображения логического времени на физическое;

— организация консистентного хранилища данных;

— тестирование полученных модулей и верификация принципов формирования структуры базы данных.

Теоретическая и практическая значимость. Результаты данного исследования могут быть применены:

— при разработке программных комплексов управления предприятиями;

— при разработке имитационных моделей производства;

— при разработке структуры консистентного хранилища данных.

Методы исследования. Для достижения поставленных задач используются следующие методы:

— реляционная алгебра;

— теория множеств;

— структурное программирование;

— модульное и интеграционное тестирование программного обеспечения.

Апробация результатов исследования. Результаты исследования используются в интегрируемом программном комплексе интеллектуальная система управления предприятием (ИСУП) – системе планирования производства (СПП). Результаты исследований по отображению логического времени на физическое были представлены в статье для Конгресса молодых ученых (2019) “Задача отображения временных промежутков на рабочий календарь”

1 Обзор существующих систем планирования производства

Индустрия 4.0 – прогнозируемое массовое внедрение киберфизических систем в промышленность и повседневный быт человека, что приведет к автоматизации бизнес-процессов предприятий. Также внедрение киберфизических систем в быт повысит качество жизни людей путем автоматизации многих повседневных циклических действий. Под киберфизической системой (CPS – cyber-physical system) подразумевается взаимодействие цифровых, аналоговых и физических компонентов, или другими словами: киберфизические системы – это интеллектуальные системы, которые состоят из тесно взаимосвязанных сетей физических и вычислительных компонентов [1].

Появилось понятие индустрии 4.0 во время ганноверской выставки 2011 года как обозначение стратегического плана развития и поддержания конкурентоспособности немецкой экономики, предусматривающий совершение прорыва в области информационных технологий для промышленности. Также считается, что данное направление знаменует собой четвертую промышленную революцию [2].

Как можно понять из названия, индустрия 4.0 в первую очередь ориентирована на промышленность. Рассматривая данную область, процесс внедрения киберфизической системы разделяется на модернизацию физической части (добавление датчиков, микроконтроллеров, и так далее...) и создание вычислительной – системы планирования производства.

Система планирования производства (СПП) обеспечивает расчет объемно-календарного и оперативного планов, автоматизированный подбор поставщиков, автоматизированный перерасчет планов по фактическим результатам деятельности, направленный на минимизацию временных и финансовых потерь.

Объемно-календарный план – задание для каждой производственной площадки на заданный интервал времени, представляющее собой план-график, на котором каждому интервалу соответствует номенклатура и объем подлежащих к производству изделий [3].

Оперативный план – план, согласно которому выполняются работы с использованием указанных производственных ресурсов.

Чтобы сохранять конкурентоспособность, предприятиям необходимо развивать и внедрять системы управления и планирования производственных процессов. Актуальным направлением, ориентированным на решение этой задачи, является разработка СПП, которая должна обеспечивать решение следующих задач: исполнения планов производства и целевых показателей, оптимизации оперативной деятельности, снижения затрат и повышения эффективности производства.

Производственное планирование — это систематический, структурированный, направленный на достижение поставленной задачи процесс планирования промышленного предприятия, состоящий из отдельных этапов, осуществляемый с помощью специальных методов и инструментов, начиная с замысла и заканчивая запуском производства [4]. Производственное планирование может также включать в себя корректирующие мероприятия в процессе эксплуатации. Планирование промышленного предприятия может основываться на различных целях и задачах, охватывать самые разные производственные ситуации.

В процессе планирования используется ряд инструментов и программных компонент, которые поддерживают реализацию метода планирования [5]. Они применяются пользователями на персональных компьютерах в различных фазах планирования промышленного предприятия. К основным инструментам планирования относятся: APS/SCM; CAP; АСУП. Ниже даны подробные описания данных типов инструментов [6].

APS/SCM (системы синхронного планирования, системы управления логической цепочкой) поддерживают расчеты, эксплуатацию и оптимизацию логистических цепочек. Особые свойства логистической цепочки получаются благодаря взаимодействию участников. Важная роль отводится структуре логистических цепочек, соответствующих рынку, а также координации и интеграции всех индивидуальных действий.

CAP (система автоматизированного регулирования) характери-

зует область компьютеризованного планирования работы. При этом используется система электронной обработки данных для создания рабочего графика, выбора эксплуатационных средств, создания указаний по изготовлению и монтажу, а также программирования для станков с ЧПУ.

АСУП (CAM — автоматическая система управления производством) включает в себя компьютерное техническое управление и контроль над производственными линиями и эксплуатационными средствами при проведении производства, т.е. прямое управление обрабатывающими и перерабатывающими машинами, устройствами манипуляции, транспортировки, перегрузки и хранения, а кратко — техническое управление всеми устройствами потоковых систем [7].

Наиболее распространенными программными продуктами, предназначенными для планирования производственных процессов являются «1С:Предприятие» и «SAP R/3»: первый является наиболее распространенным на российском рынке, второй, в свою очередь, широко используется за рубежом. На примере этих, зарекомендовавших себя с положительной стороны продуктов, проведем анализ системных компонент и сравним их место в архитектуре систем.

1.1 «1С: Предприятие 8.0»

Комплекс программ «1С: Предприятие 8.0» состоит из технологической платформы и прикладных компонентов, которые создаются на её основе и предназначены для автоматизации деятельности предприятий. Технологическая платформа не является готовым программным продуктом, предназначенным для внедрения на предприятие, вместо нее обычно применяют несколько компонентов, разработанных на её базе. Данное решение делает возможным автоматизировать различные виды деятельности, применяя единую основную технологическую платформу.

Система «1С: Предприятие 8.0» использует следующие основные компоненты:

— «Управление торговлей»;

— «Управление персоналом»;

— «Управление производственным предприятием»;

— «Управление складом»;

— «Управленческий учет и расчет себестоимости».

Наиболее интересными для рассмотрения являются компоненты «Управление персоналом» и «Управление производственным предприятием», поскольку их реализация является ключевой с точки зрения планирования деятельности предприятия и не имеет сегодня строгой математической формализации.

Компонент «1С: Предприятие 8.0. Управление персоналом» позволяет эффективно управлять кадровыми процессами в следующих областях: планирование потребностей в персонале; обеспечение организации новыми кадрами; эффективное планирование занятости персонала; кадровый учет и анализ персонала; управление персоналом.

Компонент «Управление производственным предприятием» предназначен для автоматизации процессов управления и учета на производственном предприятии. Он позволяет создать единую информационную систему для управления различными сторонами деятельности предприятия.

В платформе «1С: Предприятие 8.0» заложен ряд подходов, которые формируют основную концепцию разработки типовых компонентов. Эти подходы предназначены для максимального сближения технологических возможностей с бизнес-процессами разработки и интеграции прикладных решений. Важными моментами, которые следует отметить, являются: изоляция разработчика от технологических деталей, алгоритмическое программирование конкретной бизнес-логики приложения, использование собственной модели базы данных и гибкость прикладных решений без их доработки.

Механизм обмена данными, используемый в технологической платформе «1С: Предприятие 8.0», позволяет создавать территориально распределенные информационные системы на основе баз данных «1С:

Предприятия 8.0», и использовать другие информационных систем, не относящиеся к «1С: Предприятии 8.0». Например, можно организовать работу главного офиса, филиалов и складов предприятия в одной базе данных, или обеспечить взаимодействие базы данных «1С: Предприятия 8.0» с существующей базой данных «Oracle».

Технологическая платформа «1С: Предприятие 8.0» предоставляет средства разработки, с помощью которых создаются новые или модифицируют существующие прикладные решения. Этот инструмент разработки называется «конфигуратор». Благодаря тому, что он поставляется со стандартным пакетом «1С: Предприятия 8.0», то пользователь может свободно разработать или модифицировать прикладное решение (адаптировать его под себя), возможно, с привлечением сторонних специалистов.

Среди преимуществ данной системы можно выделить:

— открытость системы;

— регулярные программные обновления;

— широкие функциональные возможности системы.

Однако необходимо отметить, что подходы «1С: Предприятия» ориентированы на решение проблем автоматизации бухгалтерского и организационного управления предприятием. Использование проблемно-ориентированных объектов позволяет разработчику решать задачи складского, бухгалтерского, управленческого учета, расчетам заработной платы, анализа данных и управлению бизнес-процессами. Однако области экономического и бухгалтерское учета характеризуются высокой степенью математического формализации и их реализация происходит с относительно малыми трудозатратами, тогда как компоненты планирования и производственного расписания сегодня являются актуальными направлением для исследований и прикладной разработки.

1.2 «SAP R/3»

Система «SAP R/3» предоставляет собой набор разноплановых инструментов, направленных на повышение эффективности производственного процесса, увеличение экономической стабильности, автоматизацию процессов планирования. Она дает возможность интегрировать инновационные подходы централизованного планирования и управления, и повысить качество управления на разных организационных уровнях предприятий.

«SAP R/3» использует модульную архитектуру, где каждый отдельный модуль предназначен, для решения специализированной задачи процесса предприятия, взаимодействие между ними происходит в режиме реального времени. Наибольший интерес для рассмотрения представляют компоненты, не имеющие прямого отношение к бухгалтерской и экономической деятельности предприятия – модуль PP; модуль HR; модуль BC.

Модуль PP (планирование производства) дает возможность организовать управление и планирование производства предприятия. Он реализует следующие функции: формирование спецификаций, создание технологических карт, управление производственными площадками, планирование сбыта, планирование потребности в материалах, управление производственными заказами, планирование затрат на изготовление изделие, учет затрат производственных процессов, планирование производственной деятельности, управление серийным производством, планирование автоматизированного производства.

Модуль HR (управление персоналом) решает задачи планирования и управления работой персонала. Ключевые элементы: администрирование персонала, расчет данных для вычисления заработной платы, сбор и анализ данных о рабочем времени, учет командировочных расходов, создание информационной модели внутренней структуры компании.

Модуль BC (базовый модуль) предназначен для интеграции в систему «SAP R/3» всех отдельных прикладных модулей и обеспечи-

вает независимость от аппаратной платформы. Модуль BC позволяет организовать работу с многоуровневой распределенной архитектуре клиент-сервер [8].

На данный момент система «SAP R/3» является наиболее распространенной системой управления предприятием. Благодаря тому, что она является модульной системой, ее можно настроить в соответствии с конкретными потребностями отдельного предприятия. Степень технического уровня системы определяется возможностью ее перенастройки без необходимости переписывать программный код. Эта опция «SAP R/3» также позволяет занимать ведущее место в мире в системе управления.

С помощью инструментов управления, включенных в систему «SAP R/3», можно реализовывать задачи мониторинга и анализа, без дополнительного программирования, для чего система предлагает следующие способы:

— мониторинг БД;

— мониторинг операционной системы сервера;

— мониторинг коммуникаций;

— мониторинг и управление сервером приложений:

  — формирование и запуск новой конфигурация ядра R/3;

  — снятие и редактирование текущей конфигурации ядра R/3;

  — формирование временного графика в зависимости от нагрузки (например, в ночное время можно увеличивать количество процессов, отвечающих за фоновые задания);

  — управление системой архивирования;

  — управление текущими пользователями, процессами.

Многоуровневая клиент-серверная архитектура позволяет разделять задачи управления данными, ориентированные на нужды пользователя. В версии 3.0 системы «SAP R/3» SAP AG были расширены

возможности решения для организации взаимодействия с другими приложениями и распределения операций «SAP R/3» в масштабируемой компьютерной структуре. Технология внедрения «SAP R/3» основана на многоуровневой архитектуре с использованием программного обеспечения среднего уровня. Между тем, промежуточное ПО отделяет пользовательские приложения от аппаратного и программного обеспечения, с другой стороны, решает проблему взаимодействия программных приложений и аппаратного обеспечения. В «SAP R/3» SAP Basis функционирует как промежуточное программное обеспечение [8].

Основные данные — это информация, которая хранится в базе данных, достаточно долгий промежуток времени. К ним относятся такие данные как: информация о кредиторах, поставщиках, материалах и счетах. Основные данные создаются централизованно и доступны для всех приложений. Например, они включают данные клиента, которые используются в заявках, поставках, для выставления счетов и платежей и так далее. Основные данные клиента могут быть присвоены следующим организационным единицам: балансовая единица, сбытовая организация, канал сбыта, сектор.

Основная запись материала является центральным объектом данных системы «SAP R/3». Она включает в себя: сырье; оборудование; расходные материалы; полуфабрикаты; продукты; вспомогательное производственное оборудование и инструменты. Она является главным источником данных предприятия и используется всеми компонентами логистической системы SAP. Благодаря объединению всех материалов в единый объект базы данных устраняются проблемы избыточности данных. Сохранённые данные могут использоваться во всех областях, такими как закупки, контроль запасов, планирование потребностей в материалах, проверка счетов.

Данные, хранящиеся в основной записи материала, необходимы логистическому модулю системы для решения следующих задач:

— обработки запасов на поставку;

— обновления движения материалов и инвентаризационной обработки;

— проводки счетов;

— обработки клиентских заказов;

— планирования потребностей и календарного планирования.

Структурная логика поставщика и клиента также применяется к основной записи материала. При оформлении заказа для клиентов необходимо учитывать: согласование о перевозке, условия доставки, оплаты и т.д. Данные, необходимые для таких операций, дублируются из основной записи делового партнера, чтобы исключить необходимость повторного ввода информации о каждой транзакции. В основной записи материала могут одновременно храниться данные, обработанные во время ввода заказа, например, цена за единицу цены товара, запасы на другом складе и т.д. Этот принцип полезен для обработки данных в каждой основной записи, связанной с выполнением операции.

Для каждой транзакции необходимо присваивать соответствующую организационную единицу. Присвоение структуры предприятия генерируется в дополнение к данным, доступным по данным клиента и по данным материала. Поэтому документ, созданный при помощи транзакции, содержит все основные данные из организационных единиц [9].

Среди достоинств данной системы можно выделить:

— прозрачность деятельность предприятия;

— сокращение персонала управления;

— единые стандарты управления.

К недостаткам относятся:

— требуется высокий уровень подготовки персонала;

— сложность интеграции;

— большие финансовые вложения.

2 Разработка компонент системы планирования производства

2.1 Архитектура системы планирования производства

Система планирования производства представляет из себя набор программных модулей, взаимодействующих согласно схеме (рисунок 2.1).

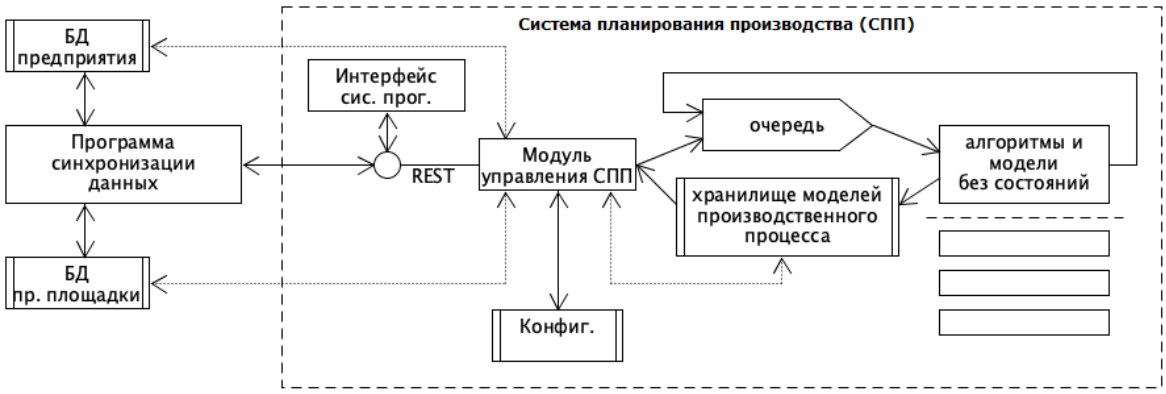

Рисунок 2.1 — Схема системы планирования производства [10]

В силу необходимости в планировании различных вариантов работы производства, каждый из которых отличается конфигурацией смен либо количеством доступных ресурсов, система производит запуск множества параллельных расчетов, по результатам которых впоследствии сотрудник сможет выбрать оптимальный. За запуск и синхронизацию отвечает “Модуль управления СПП” (рисунок 2.1), который создает очередь расчетов на запуск. Затем пул потоков (автоматическое средство для задач, которые требуют временных запусков потоков) извлекает их в порядке очереди и запускает работу подсистемы имитационного моделирования в отдельном потоке (“Алгоритмы и модели без состояний”, рисунок 2.1). После окончания работы полученный результат передается обратно в модуль управления для возврата пользователю посредством RESTful API (веб-служба, построенная с учетом архитектурного стиля REST – стиль взаимодействия компонентов распределённого приложения в сети). Пользователь, зная с какими параметрами запускался полученный расчет, может поменять конфигурацию и отправить его на повторное вычисление. Конфигура-

ция измененного расчета будет сохранена в базе данных предприятия и приведет к запуску всего цикла с начала.

2.2 Подсистема имитационного моделирования

Подсистема имитационного моделирования – модуль, производящий моделирование производственных процессов для получения приблизительной оценки времени выполнения набора операций (например карты технологического процесса).

Карта технологического процесса – документ, предназначенный для операционного описания технологического процесса изготовления или ремонта изделия (составных частей изделия) в технологической последовательности по всем операциям одного вида формообразования, обработки, сборки или ремонта с указанием переходов, технологических режимов и данных о средствах технологического оснащения, материальных и трудовых затратах (ГОСТ 3.1102-2011).

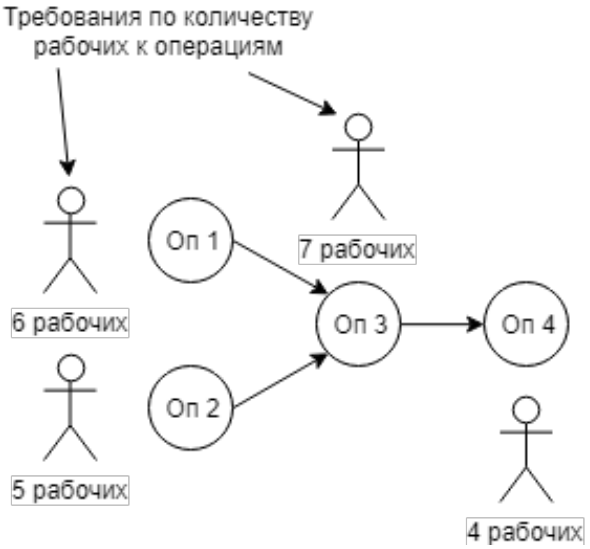

Рисунок 2.2 — Визуальное представление технологической карты с ограничением по количеству рабочих на операцию

Для имитационного моделирования подсистема преобразует карту технологического процесса в систему неравенств, в которой неизвестными являются времена начала и окончания выполнения операций:

$$\begin{cases} op1_2 = op1_1 + dur1 \\ op2_2 = op2_1 + dur2 \\ op3_2 = op3_1 + dur3 \\ op4_2 = op4_1 + dur4 \\ op1_2 \leqslant op3_1 \\ op2_2 \leqslant op3_1 \\ op3_2 \leqslant op4_1 \end{cases} \tag{2.1}$$

Здесь операция обозначается двумя переменными: отметкой начала и конца, которые обозначаются индексами 1 и 2 соответственно. Первые четыре уравнения задают расчет отметки окончания операции, другие три – накладывают ограничения на последовательность операций.

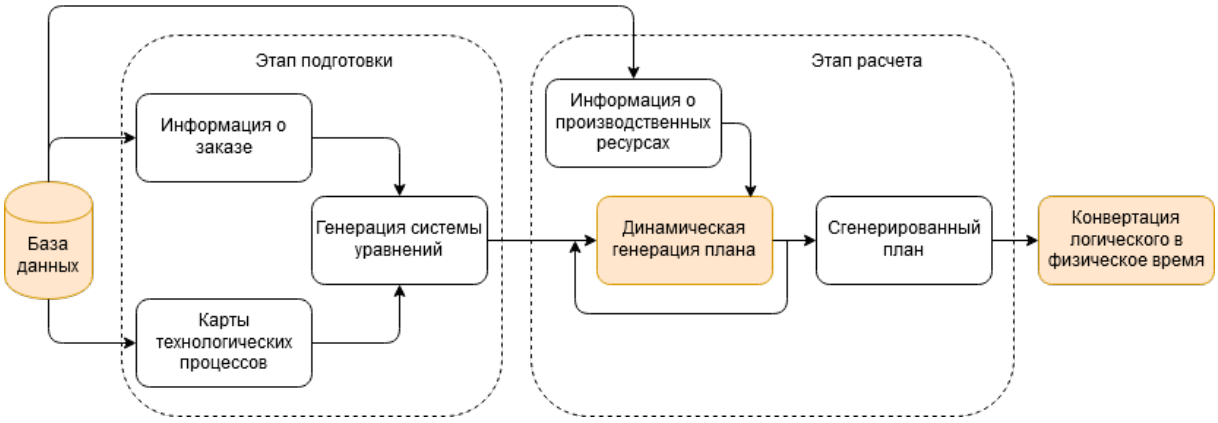

Рисунок 2.3 — Схема подсистемы имитационного моделирования

В начале работы система выберет одну из независимых переменных (переменные, которые участвуют только в одном неравенстве), например $op1_1$ или $op2_1$ из системы 2.1 – список переменных может меняться в зависимости от накладываемых моделью ресурса ограничений (о них речь в следующем разделе). Система может быть рассчитана тогда, когда можно приравнять любую из независимых переменных к логическому нулю. Тогда будет произведет подсчет времени окончания выполнения данной операции, то есть по соответствующему уравнению найдет значение окончания операции $op1_2$ или $op2_2$. Затем по тому же

принципу будет выбрано следующее уравнение и так далее пока есть неизвестные переменные. Когда они закончатся, подсистема завершит свое выполнение, передав результирующие значения и необходимые данные другому модулю, который произведет отображение полученного подсистемой имитационного моделирования результата в физическое время, о чем речь пойдет далее.

2.3 Модель ресурса сборочной линии

В процессе создания оперативного плана, для получения корректной оценки времени выполнения операции или набора операций, необходимо ввести систему ограничений для СПП, которая будет отражать влияние ресурса на её время выполнения. Это привело к созданию модели ресрсов, накладывающей ограничения на выбор операции для расчета подсистемой имитационного моделирования. Под ресурсом подразумевается любое устройство, деталь, инструмент или средство, за исключением сырьевого материала и промежуточного продукта, находящееся в распоряжении предприятия для производства товаров и услуг. В соответствии с данным определением к ресурсам относятся в том числе и человеческие ресурсы, которые в данной системе не рассматриваются с точки зрения поведения или других аспектов человеческой жизни, а лишь с точки зрения возможности выполнить конкретную задачу. Также необходимо обозначить, что в данном разделе под моделью ресурса будет пониматься упрощенная модель реального ресурса, отражающая его основные (в рамках выполняемых операций) характеристики.

Каждая модель ресурса представляет из себя структуру данных, которая должна реализовывать три метода (листинг 2.2):

— метод привязки операции к модели ресурса (Bind, строка 3 листинга 2.2);

— метод, осуществляющий проверку возможности выполнения данной операции моделью ресурса (Constrain, строка 7 листинга 2.2);

— метод, реализующий логику работы, в котором происходит изменение состояния данной модели (Done, строка 9 листинга 2.2).

Листинг 2.1 — Интерфейс ресурса

```
1   type Resource interface {
2       // Bind events to resource's logic.
3       Bind(conf interface{}, events ...*ConcreteEvent)
4       // Check resource's constrains for operation. Event must be bound.
5       // int - minimal timestamp, bool - event can be used.
6       // Constrain call can't change resource's state!
7       Constrain(event *ConcreteEvent) (int64, bool)
8       // Done this event
9       Done(event *ConcreteEvent)
10      // Clone - aux. call for copying resource.
11      Clone() Resource
12  }
```

Под привязкой операции к модели подразумевается добавление операции в очередь на выполнение и, если это первая привязанная для данного продукта операция, добавление продукта в очередь на распределение. Привязка осуществляется в начале работы системы, что позволяет ресурсам, разрешая или запрещая выбирать привязанные к ним операции для расчета, что может повлечь за собой изменение последовательности выполнения операций и, соответственно, расчетного времени выполнения карты технологического процесса (листинг 1).

Проверка производится во время работы системы и именно в этот момент происходит отбор операций в соответствии с внутренним состоянием модели (листинг 2).

Метод, реализующий логику работы вызывается при выборке операции подсистемой и для каждой вызывается два раза: чтобы отметить состояние модели в начале и в конце расчета операции (см. рисунок 2.4, листинг 3).

СПП имеет несколько видов ресурсов, одним из которых является модель ресурса сборочной линии. Она описывает несколько однотипных, то есть с одинаковым числом рабочих постов, физических сборочных линий. Сборочная линия – это способ перемещения заготовки от одного рабочего поста к другому; на каждом посту выполняются

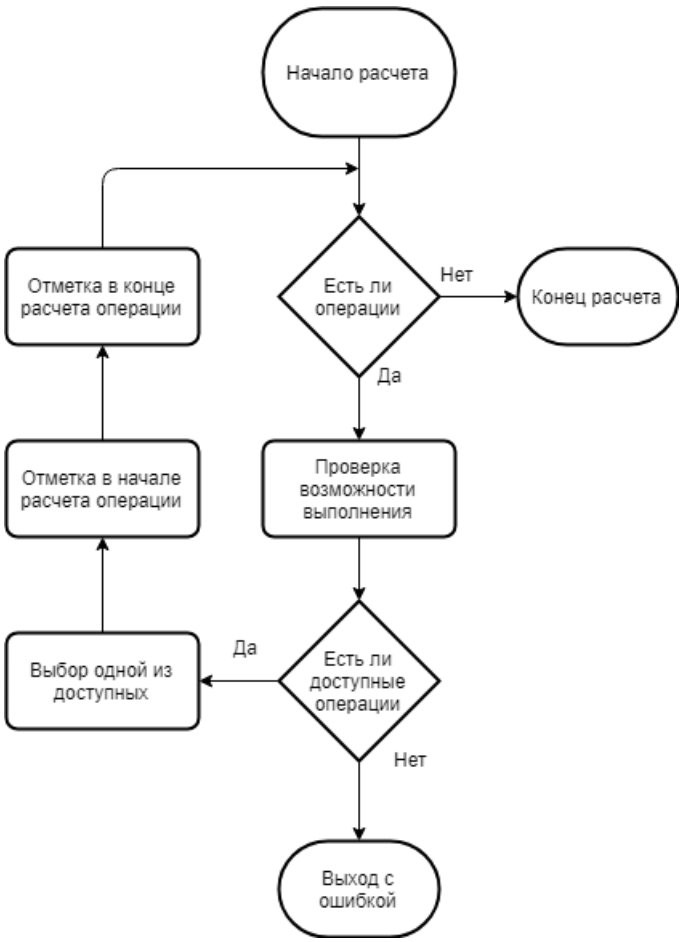

Рисунок 2.4 — Схема работы подсистемы моделирования с ресурсами в процессе расчета оперативного плана

закрепленные за ним операции. Под постами понимаются заготовко-места, оснащенные соответствующим технологическим оборудовани-ем и предназначенными для технического воздействия на заготовку для осуществления фиксированного перечня операций. Объединение нескольких сборочных линий в одну обуславливается упрощением как взаимодействия с подсистемой имитационного моделирования, так и управления моделью, потому как в любом случае (даже когда все ли-нии будут различны по количеству постов) количество линий в модели будет всегда меньше либо равно количеству физических сборочных линии. Благодаря такому объединению представляется возможным ин-капсулировать реализацию распределения заготовок и связанных с ними операций по сборочным линиям внутри модели, а также опи-сывать ситуации, когда один и тот же рабочий может перемещаться между линиями в пределах одного поста.

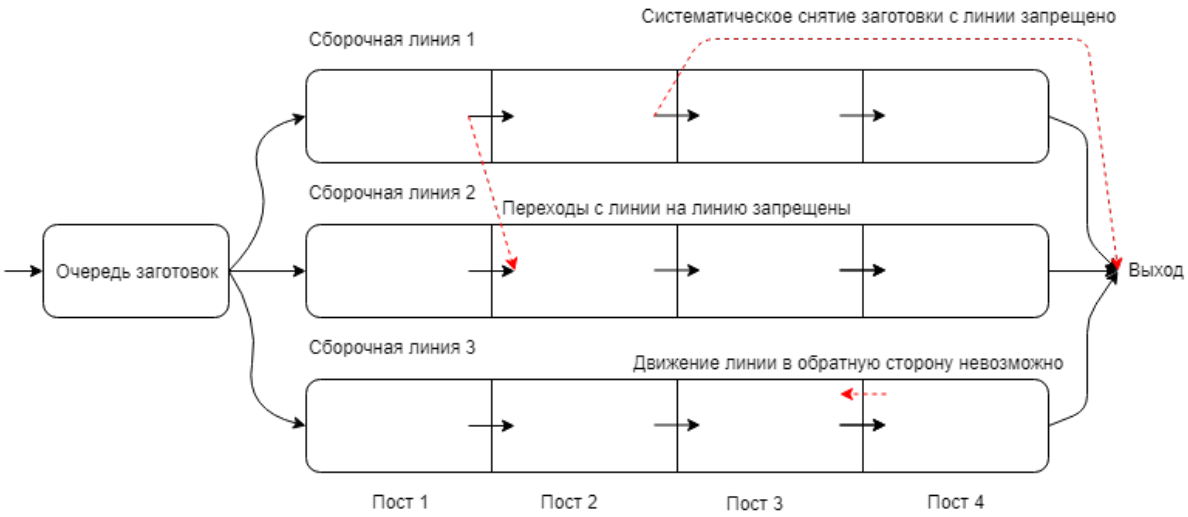

Рисунок 2.5 — Схема модели ресурса с вариантами перемещения заготовки внутри

Главным назначением модели, как и реальной сборочной линии, является ограничение перемещения продукции внутри ресурса (см. рисунок 2.5), и моделирование работы физического сборочного конвейера. С одной стороны ограничивается перемещение между сборочными линиями: к какой продукт был привязан, на той он и останется до окончания выполнения всех операций, относящихся к данному продукту и привязанных к постам данной сборочной линии. С другой – перемещение продукции между рабочими постами: продукт должен двигаться последовательно с поста на пост (см. рисунок 2.5).

Листинг 2.2 — Структуры сборочной линии

```
1   // Describes one product.
2   type Product struct {
3       TypeName      imcore.ProductTypeName
4       SerialNumber  imcore.ProductSN
5   }
6
7   // Describes one station
8   type workstation struct {
9       isWorking bool       // Is station working?
10      isEmpty   bool       // Is station empty? (if station not working
            this
11                           // doesn't mean that station is empty).
12      time      int64      // Current timestamp of this station.
```

```
13          product   Product   // Product on this station.
14      }
15
16      // Describes one line.
17      type productionLine struct {
18          stations []workstation
19          maxTime  int64
20          minTime  int64
21      }
22
23      // AssemblyLine – assembly line resource.
24      //  Contains *NumberOfLines* lines that
25      //  which consist of same *NumberOfStations*.
26      type AssemblyLine struct {
27          // NumberOfStations – number of stations for each line.
28          NumberOfStations int
29          // NumberOfLines – number of lines of this resource.
30          NumberOfLines int
31          // Operations queue.
32          remainEvents []map[Product][]*imcore.ConcreteEvent
33          lines        []productionLine // resource lines.
34          queue        []Product        // Production queue.
35          remain       map[*imcore.ConcreteEvent][]*imcore.ConcreteEvent
36          complete     map[*imcore.ConcreteEvent][]*imcore.ConcreteEvent
37      }
```

Одной из ключевых особенностей практически любой сборочной линии является синхронизация передвижения продукции между постами. Это означает что, перемещение продукции на сборочной линии будет осуществляться с периодом, равным максимальной длительности выполнения всех операций на постах – эта длительность называется тактом сборочной линии. Каждая заготовка сможет сменить пост только после того, как все остальные заготовки будут готовы к смене своих постов.

Для реализации необходимого функционала, были введены структуры, описывающие линии, посты, очередь продукции, и очередь операций на каждый пост всех линий. Каждая линия, моделируемая компонентом, характеризуется временем начала производственного цикла, структурой данный, описывающей набор рабочих постов (в свою очередь описываемые состоянием: "выполняются работы "простаива-

ет "отсутствует продукция на посту технической картой и серийным номером заготовки).

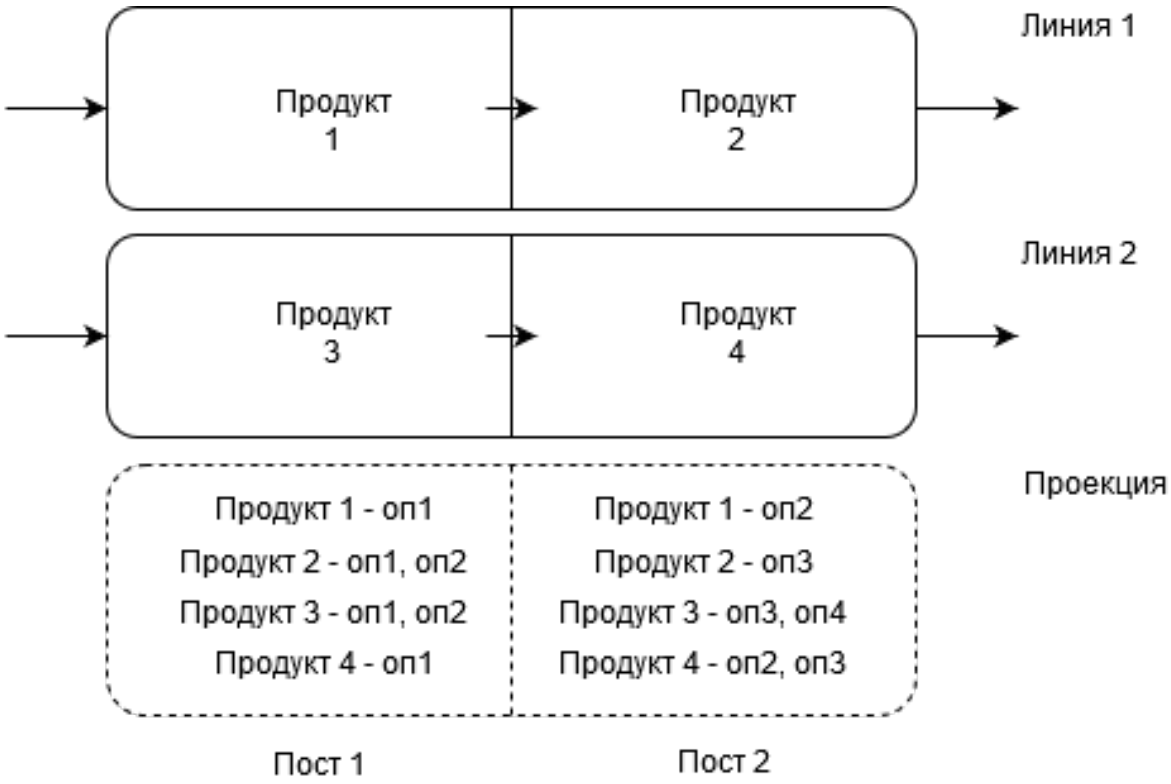

Рисунок 2.6 — Схема сборочной линии с очередью операций на посты

Очередь операций на каждый пост – структура данных необходимая для динамической привязки операций к линиям модели. Так как во время привязки операции система не может оценить длительность изготовления продукции, а распределение заготовок происходит до начала работы, может сложиться ситуация, когда одна из линий будет работать намного дольше или меньше по сравнению с другими линиями, что приведет к простою производства. Следовательно необходимо, не привязывая операцию к определенной линии, обозначить к какому посту она относится (см. рисунок 2.6). Это является основной задачей разработанной структуры – она содержит то же количество постов, что и остальные линии, но не описывает какую либо физическую сборочную линию, а является очередью операций всех постом для всех линий. Внутри каждого поста данной очереди находится хэш-таблица[1],

[1]структура данных, позволяющая хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу

в которой ключом является конкретная единица продукции (характеризующаяся типом продукции и серийным номером), а значением – список операций, который необходимо выполнить над данной заготовкой на посту.

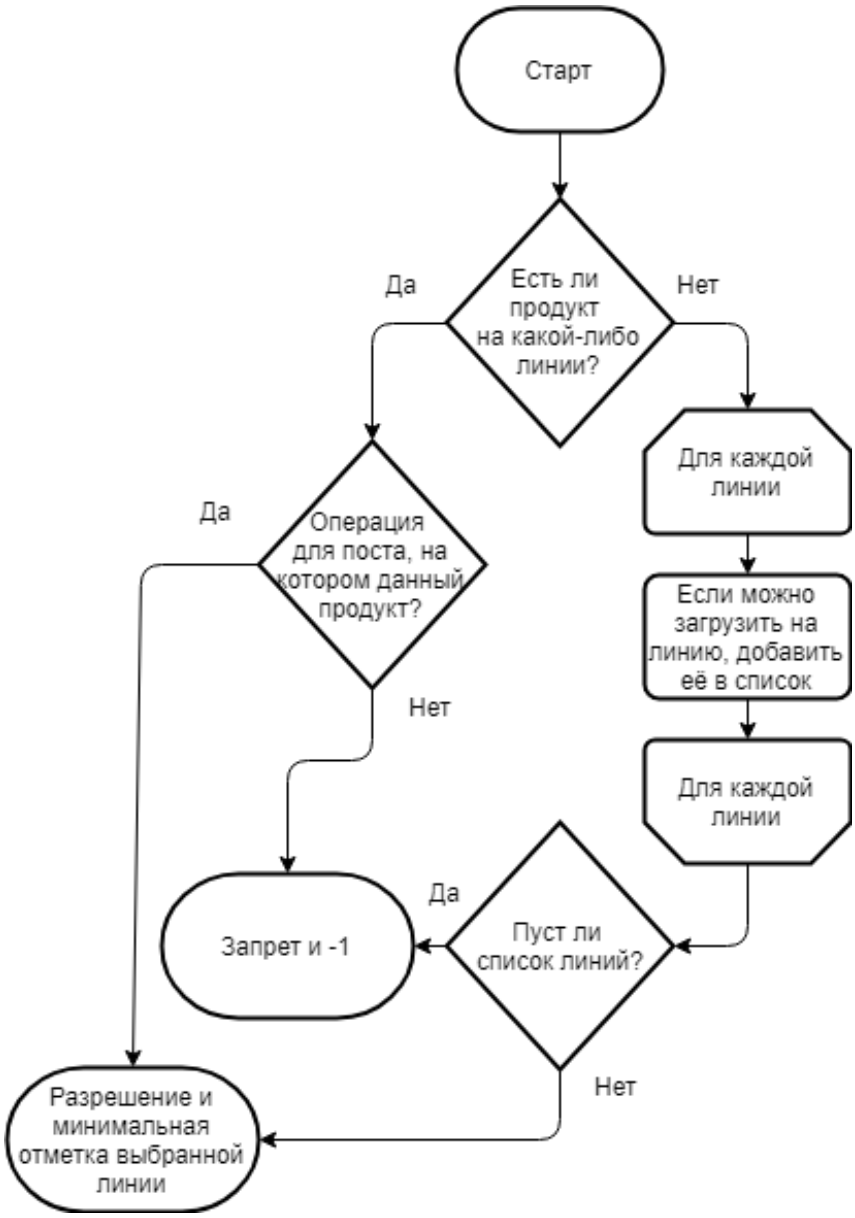

Рисунок 2.7 — Схема проверки возможности выполнения операции

Во время выполнения система, для определения возможности выполнения операции опрашивает все линии с целью определения возможности выполнения данной операции и местонахождения продукта (рисунок 2.7, листинг 2):

— при отсутствии продукта на всех линиях:

1) инициируется проверка всех линий на возможность загрузки первого поста (то есть пуст ли он, листинг 8);

2) из получившегося списка линий (если количество больше нуля, иначе переход к 4 пункту) выбирается линия, временная отметка которой является наименьшей среди всех доступных линий;

3) возвращается разрешение на выполнение данной операции и временная метка, выбранная на предыдущем шаге;

4) иначе (количество линий равно нулю) возвращается запрет на выполнение данной операции на текущей итерации.

— если продукт находится на какой-либо линии (листинг 7), то производится опрос очереди операций:

1) при нахождении нужной операции на посту, на котором находится в данный момент продукт, возвращается разрешение на выполнение и временная метка, с которой может производиться данная операция;

2) иначе возвращается запрет выполнения.

Подсистема имитационного моделирования последовательно переберет все доступные на данной итерации операции и выберет одну из тех, что получили разрешение от всех моделей ресурсов на выполнение. Если таковых не будет, то система известит о невозможности дальнейшей работы вследствие логической ошибки во входных данных. В ином случае произойдет вызов метода для того, чтобы отметить состояние модели в начале выполнения операции. При этом, если продукт не был загружен на линию, то он будет загружен, иначе произойдет проверка на возможность сдвига линии.

Во второй раз метод будет вызван для того, чтобы отметить окончание операции – это означает что данная операция будет удалена из проекции, временная метка поста будет изменена с учетом длительности операции, и, если операций на данном посту для данного

продукта не осталось, то состояние поста изменится на “готов к сдвигу линии” и будет совершена проверка возможности сдвига линии.

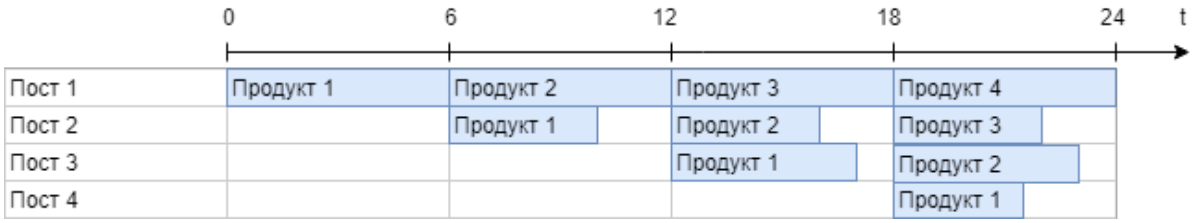

Рисунок 2.8 — Диаграмма, изображающая сдвиг линии

Сдвиг линии – процесс, при котором все посты на сборочной линии передают свою заготовку на следующий пост и принимают заготовку с предыдущего. Данный процесс происходит одновременно для всех постов, не может быть разделен или проигнорирован каким-либо постом и выполняется после получения от всех постов сигнала о готовности к сдвигу. При этом происходит синхронизация постов, то есть все посты, и, соответственно вся линия, получают одну временную отметку – сумма предыдущей отметки начала рабочего такта и длительности текущего рабочего такта линии. С этой отметки начинается отсчет следующего такта (см. 2.8, листинг 5).

Листинг 2.3 — Интерфейс ресурса

```
2019/05/31 23:10:35 Line (1) product (M, 1) loaded on station 0
2019/05/31 23:10:35 Line (1) product (M, 1) changing timestamp (4)
2019/05/31 23:10:35 Line (1) product (M, 1) changing station (0 -> 1)
2019/05/31 23:10:35 Line (1) product (M, 1) changing timestamp (7)
2019/05/31 23:10:35 Line (2) product (M, 2) loaded on station 0
2019/05/31 23:10:35 Line (2) product (M, 2) changing timestamp (4)
2019/05/31 23:10:35 Line (2) product (M, 2) changing station (0 -> 1)
2019/05/31 23:10:35 Line (2) product (M, 2) changing timestamp (7)
2019/05/31 23:10:35 Line (1) product (M, 3) loaded on station 0
2019/05/31 23:10:35 Line (1) product (M, 3) changing timestamp (8)
2019/05/31 23:10:35 Line (1) product (M, 1) changing station (1 -> 2)
2019/05/31 23:10:35 Line (1) product (M, 3) changing station (0 -> 1)
2019/05/31 23:10:35 Line (1) product (M, 1) changing timestamp (14)
2019/05/31 23:10:35 Line (2) product (M, 2) changing station (1 -> 2)
2019/05/31 23:10:35 Line (2) product (M, 2) changing timestamp (13)
2019/05/31 23:10:35 Line (2) product (M, 2) unloaded from line
    (timestamp: 13)
2019/05/31 23:10:35 Line (1) product (M, 3) changing timestamp (11)
```

```
18      2019/05/31 23:10:35 Line (1) product (M, 1) unloaded from line
            (timestamp: 14)
19      2019/05/31 23:10:35 Line (1) product (M, 3) changing station (1 -> 2)
20      2019/05/31 23:10:35 Line (1) product (M, 3) changing timestamp (20)
21      2019/05/31 23:10:35 Line (1) product (M, 3) unloaded from line
            (timestamp: 20)
```

Для наглядного представления процессов происходящих в данной модели, на каждой итерации работы создается несколько записей в журнал для отслеживания состояния модели в данный момент – по ним можно отследить некорректное поведение, либо просто узнать причины, по которым подсистема имитационного моделирования отработала именно в данной последовательности (естественно при условии работы с моделью ресурса сборочной линии). На листинге 2.3 видно, что модель ресурса охватывает две линии (1 и 2) физических конвейеров и работает с тремя продуктами типа “M” (1, 2 и 3). Также в данном журнале видны все передвижения продуктов между постами (station), все изменения временных меток и результирующие метки отгрузки с последних постов.

По завершении основной разработки, было произведено как ручное тестирование (просмотром логов в поисках ошибок работы, листинг 2.3), так и автоматизированное с написанием модульных (для проверки самой модели) и интеграционных тестов (для проверки работы с подсистемой имитационного моделирования и другими моделями ресурсов) – которые показали правильную работу во всех рассмотренных случаях.

В результате проведенного тестирования (25 тестов), покрытие кода составило 84.3%, что отражено в листинге 2.4.

Листинг 2.4 — Тестовое покрытие кода

```
1   PASS
2   coverage: 84.3% of statements
3   ok  nitta.io/yamp/imcore/resource   1.041s  coverage: 84.3% of
        statements
4   Success: Tests passed.
```

2.4 Модуль отображения логического времени на физическое

Так как расчет выполнения операции (как и всей карты технологического процесса) подсистемой имитационного моделирования производится в логическом времени, то есть во времени отсчитываемом от нуля, существует необходимость в отображении (соответствие между элементами двух множеств) логического времени на физическое. Одной из главных сложностей, возникающих при этом, является неоднородность рабочего времени, которая проявляется в рабочем графике (чередование интервалов рабочего и нерабочего времени), наличии выходных и перенесенных дней. Другой сложностью является наличие в системе “обратного расчета”, при котором планирование ведется от даты “дедлайна” (дата или время, к которому должна быть выполнена задача), что накладывает свои ограничения на реализацию данной компоненты, такие как:

— проблемы с определением рабочих интервалов, которые относятся к текущему дню;

— во время обратного расчета происходит движение в другую сторону по оси физического времени;

— смещение интервалов рабочего времени относительно рабочего дня.

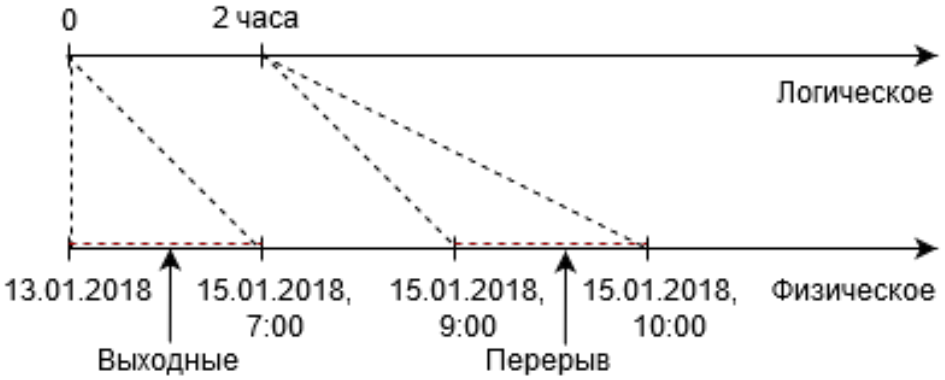

Рисунок 2.9 — Пример отображения оси логического на ось физического времени

Для отображения логического времени на физическое был предложен итеративный процесс, который осуществляет “переход” к необходимому времени путем последовательного перебора дат.

Как было сказано ранее, из-за того, что рабочее время не является непрерывным, мы не имеем возможности просуммировать начальную дату и значение поданного логического времени. Это ведет к тому, что необходимо синхронизировать логическое и физическое время – это достигается путем последовательного периодического отображения конкретного логического времени на физическое (см. рисунок 2.9). Это подразумевает под собой наличие двух массивов чисел или “осей”:

— оси логического времени, которая начинается с нуля и единица которой соответствует одной секунде (необходимости в более точном отображении нет);

— оси физического времени, на которой может быть отложено любая дата физического времени, отсчет которой начинается 1 января 1970 года 00:00:00 (эпоха Unix).

Особенностью оси физического времени является наличие на ней “выколотых” точек – промежутков времени в которые работа не ведется и операции не выполняются.

Входными данными для модуля являются:

— дата, с которой необходимо начинать отсчет;

— логическое время, которого необходимо достигнуть;

— конфигурация, состоящая из данных о рабочем графике, о датах выходных и перенесенных дней.

Дата является точкой на физической оси, куда будет отображаться нуль логической. Она представляет собой количество секунд, прошедшее с начала эпохи Unix.

Логическое время – количество секунд, которое должно быть отложено на логической оси. В силу непрерывности физической оси,

каждой логической точке сопоставляется отрезок на физической оси, сопоставляется пара чисел – границ данного отрезка.

Листинг 2.5 — Структуры конфигурации

```
1   // WorkInterval : one work interval.
2   type WorkInterval struct {
3       ShiftID int
4       Starts  int64
5       Ends    int64
6   }
7
8   // DayTemplateName : name for day template.
9   type DayTemplateName string
10
11  // ScheduleForDay : work intervals for days.
12  type ScheduleForDay map[DayTemplateName][]WorkInterval
13
14  // ScheduleTemplate : schedule template for week.
15  type ScheduleTemplate [7]DayTemplateName
16
17  // ExceptionDates : exception dates for year.
18  type ExceptionDates map[int64]DayTemplateName
19
20  // ConverterConfig : information for mapping logical time to physical.
21  type ConverterConfig struct {
22      // Weekly schedule template.
23      ScheduleTemplate ScheduleTemplate
24      // Exception dates for year.
25      ExceptionDates ExceptionDates
26      // Work intervals for every day.
27      ScheduleForDay ScheduleForDay
28  }
```

Конфигурация модуля – данные используемые для определения модулем какие промежутки являются выколотыми точками на оси физического времени. Состоит из данных о рабочем графике занятого персонала (интервалы рабочего времени), шаблонном расписании на неделю (например, суббота, воскресенье – выходные, пятница – “короткий” день, остальные – стандартные рабочие дни), набора информации о датах, которые являются днями-исключениями и соответствующей информацией о графике работы в данные дни.

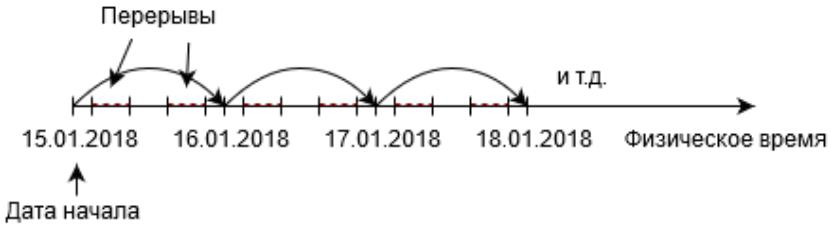

Рисунок 2.10 — Схема прямого расчета

Рисунок 2.11 — Схема обратного расчета

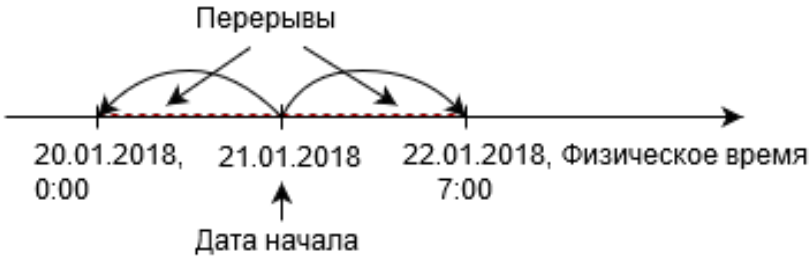

Рисунок 2.12 — Схема проверки времени

После запуска модуль получает параметры и совершает проверку последних на корректность и непротиворечивость (например, если два дня имеют пересечения временных промежутков, то они противоречивы, ведь ресурс не может работать одновременно в двух сменах) как в рамках смен одного, так соседних дней. Далее производится определение режима работы: прямой, обратный расчет или проверка времени:

— прямой расчет – задается дата начала отсчета, логическое время и расчет ведется до нахождения даты окончания работ (см рисунок 2.10);

— обратный расчет – задается дата дедлайна, логическое время и расчет ведется до нахождения времени начала работ (см рисунок 2.11);

— проверка времени – задается дата и логическое время равное нулю, что запускает оба предыдущих расчета пока не будет найдено первое ненулевое время в обоих направлениях от даты начала расчета (см рисунок 2.12).

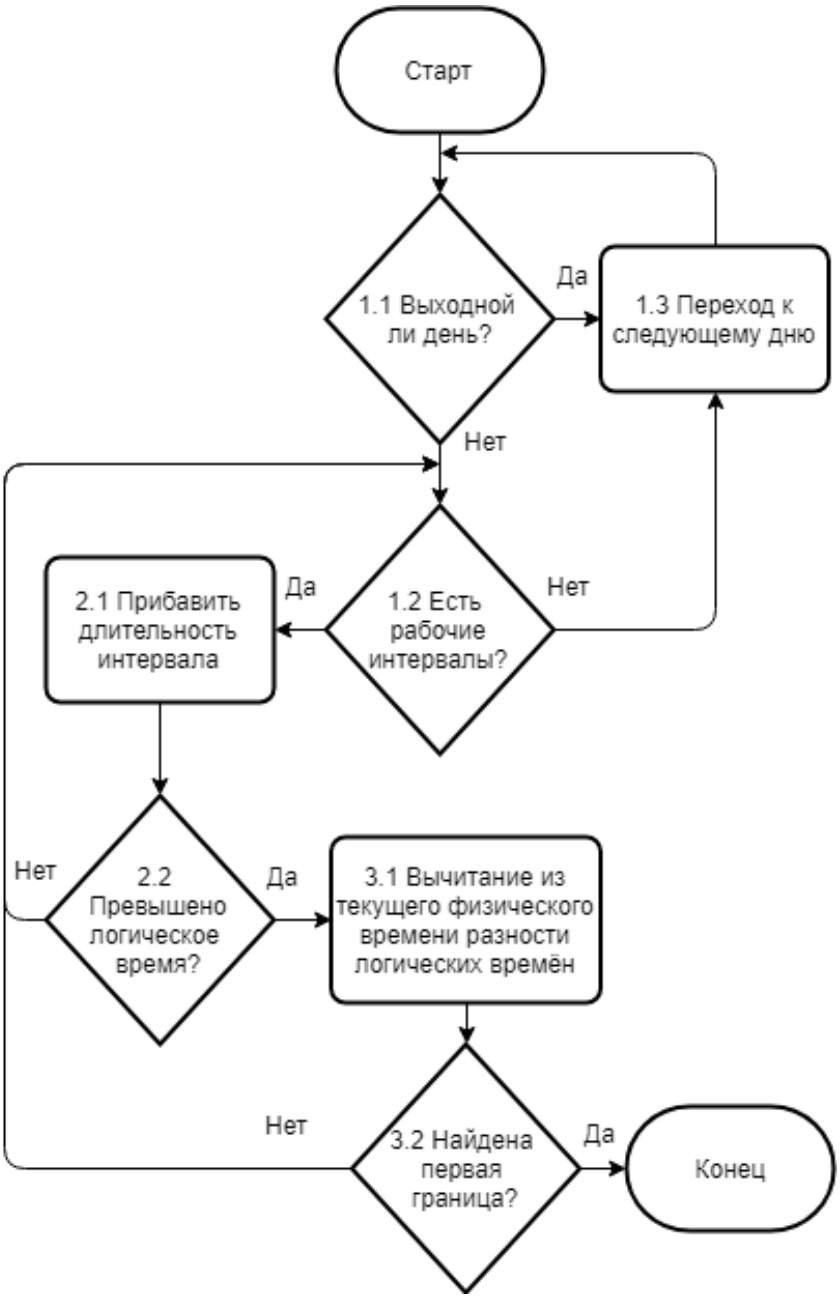

Рисунок 2.13 — Блок-схема модуля

После выбора режима работы счетчик текущего логического времени обнуляется и счетчик текущего физического времени до стартовой даты. Затем итеративно, пока текущее логическое время не превысит необходимое производится поиск следующей даты. Алгоритмически поиск даты работает следующим образом (рисунок 2.13, листинг 11):

1) определяются интервалы рабочих смен относящихся к текущему дню (рисунок 2.13, п. 1.1, листинг 13): при отсутствии таковых (рисунок 2.13, п. 1.2), к текущей дате прибавляется один день и затем возврат к п.1 (рисунок 2.13, п. 1.3).

2) отсортированные в порядке возрастания интервалы последовательно перебираются и их длительности прибавляются к текущему логическому и физическому времёнам (рисунок 2.13, п. 2.1, листинг 12):

а) при превышении текущим логическим временем необходимого, переход к п.3 (рисунок 2.13, п. 2.2);

б) если все интервалы были просуммированы, но необходимое логическое время не превышено – переход к п.1 (рисунок 2.13, п. 2.2);

3) разность текущего и необходимого логического времён вычитается из физического времени (рисунок 2.13, п. 3.1), при этом сохраняя данное значение как левую (правую при обратном расчете) границу, после чего продолжается расчет для выявления правой (левой) границы промежутка (рисунок 2.13, п. 3.2).

Определение интервалов рабочего времени происходит взятием даты из текущего физического времени (листинг 15) – затем начинается определение принадлежности данной даты к перенесенным датам после чего есть два варианта:

— дата является перенесенным днем и модуль получает информацию о расписании которое нужно применить;

— дата не является перенесенным днем и получение информации происходит исходя из того, каким днем недели является данная дата.

На предприятиях нередко случается так, что рабочие интервалы, принадлежащие к рабочему дню и сам рабочий день смещены относительно друг друга (см. рисунок 2.14). При этом возникают трудности с определением интервалов рабочего времени в связи с тем, что для опре-

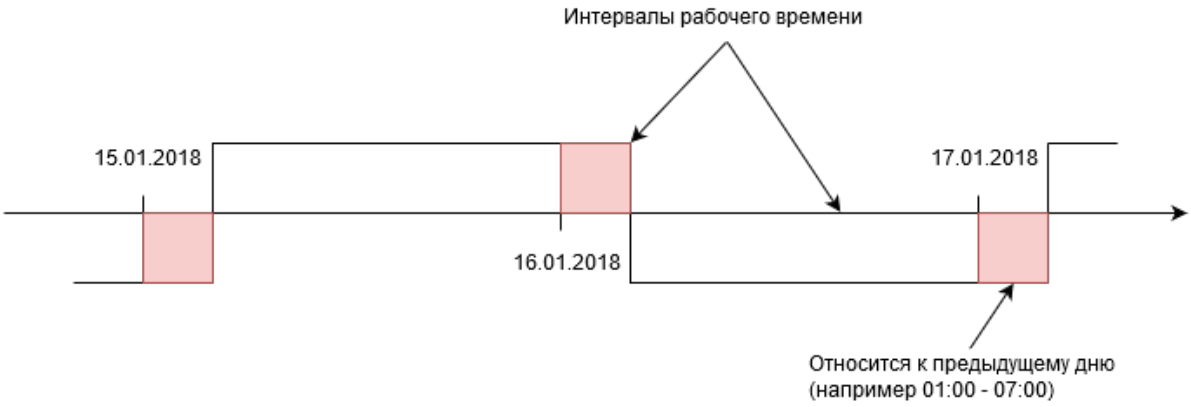

Рисунок 2.14 — Смещение интервалов рабочего времени относительно рабочего дня

Рисунок 2.15 — Граница двух дней

деления используется количество секунд с начала эпохи Unix и до нуля часов нуля минут и нуля секунд нужной даты. По количеству секунд, используя инструментарий языка, определяется каким днем недели является нужная дата и, соответственно, по дню недели задается шаблон рабочего дня. Эти трудности практически не влияют на прямой расчет, но с обратным все немного сложнее. Так как в одном дне 86400 секунд и, если рассматривать границу двух дней (см. рисунок 2.15), то 86400 секунда предыдущего дня будет равна нулевой секунде нового дня. Можно сказать, что 86400 – левый предел, а 0 – правый предел в данной точке (особенно, если изображать в виде окружности). Но, в связи с тем, что данная недетерминированность присутствует лишь при рассмотрении ситуации человеком, а система может распознавать диапазон от 0 до 86399 секунд. Тогда, в качестве решения данной проблемы, для определения интервалов рабочего графика (при обратном расчете) используется величина в 86399 секунд, как последняя секунда текущего дня, при условии, что данное допущение не влияет на расчет.

Листинг 2.6 — Пример конфигурации

```
2019/05/31 16:01:23 New config 1: {
    ScheduleTemplate: {
        Sunday    :  Day Off,
        Monday    :  Workday,
        Tuesday   :  Workday,
        Wednesday:  Workday,
        Thursday  :  Workday,
        Friday    :  Shortday,
        Saturday  :  Day Off,
    },
    ExceptionDates:{
        1514764800 (01/01/2018 00:00:00): Day Off,
        1514851200 (02/01/2018 00:00:00): Day Off,
        1514937600 (03/01/2018 00:00:00): Day Off,
        ...
        1515888000 (14/01/2018 00:00:00): Day Off,
    },
    ScheduleForDay:{
        Day Off : {
        }
        Shortday : {
            { ShiftID: 2, Starts: 28800  (01 day 08:00:00), Ends:
                46800  (01 day 13:00:00) }
            { ShiftID: 2, Starts: 50400  (01 day 14:00:00), Ends:
                64800  (01 day 18:00:00) }
        }
        Workday : {
            { ShiftID: 1, Starts: 28800  (01 day 08:00:00), Ends:
                46800  (01 day 13:00:00) }
            { ShiftID: 1, Starts: 50400  (01 day 14:00:00), Ends:
                64800  (01 day 18:00:00) }
            { ShiftID: 1, Starts: 68400  (01 day 19:00:00), Ends:
                86400  (02 day 00:00:00) }
            { ShiftID: 1, Starts: 90000  (02 day 01:00:00), Ends:
                111600 (02 day 07:00:00) }
        }
    }
}
```

Возвращаясь к пункту 3 алгоритма при нахождении нужного времени работа модуля не прекращается, а ведется до момента нахождения второй границы промежутка, на который отображается необходимое логической время.

Как было сказано ранее – физическое время непрерывно, а следовательно когда производится отображение на него логического времени, в результате получается промежуток (см. рисунок 2.9), который и характеризуют две границы. Эта пара чисел, характеризующая начало и конец отрезка, которые отображаются на логическую ось в точке, где значение равно входному логическому времени и являются выходными данными данного модуля.

Листинг 2.7 — Пример прямого расчета модулем

```
2019/05/31 15:10:50 Start time is: 1514678400 (31/12/2017 00:00:00)
2019/05/31 15:10:50 Given logical time is: 518400 (144 in hours)
2019/05/31 15:10:50  Logical time | Physical time
2019/05/31 15:10:50 ——————————————|————————————————————————————
2019/05/31 15:10:50  0            | 1515974400 (15/01/2018 00:00:00)
2019/05/31 15:10:50  18000        | 1516021200 (15/01/2018 13:00:00)
2019/05/31 15:10:50  32400        | 1516039200 (15/01/2018 18:00:00)
2019/05/31 15:10:50  50400        | 1516060800 (16/01/2018 00:00:00)
2019/05/31 15:10:50  72000        | 1516086000 (16/01/2018 07:00:00)
2019/05/31 15:10:50  90000        | 1516107600 (16/01/2018 13:00:00)
2019/05/31 15:10:50  104400       | 1516125600 (16/01/2018 18:00:00)
2019/05/31 15:10:50  122400       | 1516147200 (17/01/2018 00:00:00)
...
2019/05/31 15:10:50  424800       | 1516730400 (23/01/2018 18:00:00)
2019/05/31 15:10:50  442800       | 1516752000 (24/01/2018 00:00:00)
2019/05/31 15:10:50  464400       | 1516777200 (24/01/2018 07:00:00)
2019/05/31 15:10:50  482400       | 1516798800 (24/01/2018 13:00:00)
2019/05/31 15:10:50  496800       | 1516816800 (24/01/2018 18:00:00)
2019/05/31 15:10:50  514800       | 1516838400 (25/01/2018 00:00:00)
2019/05/31 15:10:50  518400       | 1516845600 (25/01/2018 02:00:00)
2019/05/31 15:10:50 Result:
2019/05/31 15:10:50 ————————————————————————————————————————————————
—— || —— 518400 | [1516845600          – 1516845600          ]
—— || —— 518400 | [25/01/2018 02:00:00 – 25/01/2018 02:00:00]
——— PASS: TestOffsetToTimeLine (0.08s)
PASS
ok      nitta.io/yamp/schedule  0.860s
Success: Tests passed.
```

В результате работы над модулем было проведено тестирование, результаты одного из которых можно увидеть на листингах 2.6 и 2.7. На листинге 2.6 приведена конфигурация модуля, которая показывает, как

производится совмещение логического и физического дня, обработка перенесенных дней. На листинге 2.7, перед расчетом нового отображения, можно отметить, что при использовании той же конфигурации не производится ее повторный вывод в журнал, что позволяет сократить его длину, а следовательно улучшить читаемость. Листинг 2.7 показывает итеративный процесс как результат прибавления каждого нового интервала к текущему времени.

Листинг 2.8 — Пример обратного расчета модулем

```
2019/05/31 15:09:35 Start time is: 1516845600 (25/01/2018 02:00:00)
2019/05/31 15:09:35 Given logical time is: −518400 (−144 in hours)
2019/05/31 15:09:35  Logical time | Physical time
2019/05/31 15:09:35 ——————————————|————————————————————————
2019/05/31 15:09:35  0            | 1516845600 (25/01/2018 02:00:00)
2019/05/31 15:09:35  3600         | 1516838400 (25/01/2018 00:00:00)
2019/05/31 15:09:35  21600        | 1516820400 (24/01/2018 19:00:00)
2019/05/31 15:09:35  36000        | 1516802400 (24/01/2018 14:00:00)
2019/05/31 15:09:35  54000        | 1516780800 (24/01/2018 08:00:00)
2019/05/31 15:09:35  75600        | 1516752000 (24/01/2018 00:00:00)
2019/05/31 15:09:35  93600        | 1516734000 (23/01/2018 19:00:00)
2019/05/31 15:09:35  108000       | 1516716000 (23/01/2018 14:00:00)
2019/05/31 15:09:35  126000       | 1516694400 (23/01/2018 08:00:00)
...
2019/05/31 15:09:35  414000       | 1516129200 (16/01/2018 19:00:00)
2019/05/31 15:09:35  428400       | 1516111200 (16/01/2018 14:00:00)
2019/05/31 15:09:35  446400       | 1516089600 (16/01/2018 08:00:00)
2019/05/31 15:09:35  468000       | 1516060800 (16/01/2018 00:00:00)
2019/05/31 15:09:35  486000       | 1516042800 (15/01/2018 19:00:00)
2019/05/31 15:09:35  500400       | 1516024800 (15/01/2018 14:00:00)
2019/05/31 15:09:35  518400       | 1514592000 (30/12/2017 00:00:00)
2019/05/31 15:09:35 Result:
2019/05/31 15:09:35
    ————————————————————————————————————————————————————————————
-- || -- 518400 | [1516003200          − 1514570400          ]
-- || -- 518400 | [15/01/2018 08:00:00 − 29/12/2017 18:00:00]
--- PASS: TestOffsetToTimeLineReverted (0.06s)
PASS
ok      nitta.io/yamp/schedule  0.670s
Success: Tests passed.
```

Также были проведены тесты обратного расчета и проверки времени, результаты которых можно видеть на листингах 2.8 и 2.9. Пояснение к анализу заданного физического времени: используется

конфигурация, где 2 января является выходным днем, анализ которого и производится, что позволяет продемонстрировать работу данного решения.

Листинг 2.9 — Пример анализа заданного физического времени

```
    2019/05/31 15:12:07 Start time is: 1515326580 (07/01/2018 12:03:00)
    2019/05/31 15:12:07 Given logical time is: 0 (0 in hours)
    2019/05/31 15:12:07  Logical time | Physical time
    2019/05/31 15:12:07 ——————————————|————————————————————————————
    2019/05/31 15:12:07  0            | 1515196800 (06/01/2018 00:00:00)
    2019/05/31 15:12:07  0            | 1515196800 (06/01/2018 00:00:00)
    2019/05/31 15:12:07 Result:
    2019/05/31 15:12:07
       ————————————————————————————————————————————————————————————
    —— || —— 0     | [1515196800          – 1515196800          ]
    —— || —— 0     | [06/01/2018 00:00:00 – 06/01/2018 00:00:00]
    2019/05/31 15:12:07  Logical time | Physical time
    2019/05/31 15:12:07 ——————————————|————————————————————————————
    —— || —— 0           | 1515369600 (08/01/2018 00:00:00)
    —— || —— 0           | 1515369600 (08/01/2018 00:00:00)
    2019/05/31 15:12:07 Result:
    2019/05/31 15:12:07
       ————————————————————————————————————————————————————————————
    —— || —— 0     | [1515369600          – 1515369600          ]
    —— || —— 0     | [08/01/2018 00:00:00 – 08/01/2018 00:00:00]
    ——— PASS: TestStartOnWeekendWithZero (0.06s)
    PASS
    ok      nitta.io/yamp/schedule  0.594s
    Success: Tests passed.
```

В результате проведенного тестирования (40 тестов), покрытие кода составило 96.6%, что отражено в листинге 2.10.

Листинг 2.10 — Тестовое покрытие кода

```
    PASS
    coverage: 96.6% of statements
    ok      nitta.io/yamp/schedule  0.974s  coverage: 96.6% of statements
    Success: Tests passed.
```

3 Организация консистентного хранилища данных

3.1 Выбор хранилища данных

Система планирования производства, как и практически любая система, получающая и обрабатывающая данные, нуждается в хранилище данных – базе данных. Хранилище данных может быть разделено на две составляющие:

— хранилище временных данных;

— хранилище постоянных данных.

Под временными данными подразумеваются данные, которые получаются в процессе работы системы (например оперативные и объемно-календарные планы) и, при необходимости, могут быть рассчитаны заново, хоть и с некоторыми затратами (время или вычислительные мощности). В данной работе этот вид данных и хранилище для них не рассматривается.

С другой стороны существуют постоянные данные – информация которая задается, например предприятием, потеря которой в лучшем случае приведет к необходимости заново добавлять их в систему, а в худшем – приведет к утрате данной информации. В любом случае потеря постоянных данных ведет к критическим нарушениям в работе системы, что обуславливает необходимость организации консистентного хранилища данных.

Консистентность (англ. data consistency, согласованность) данных– требование к данным, получаемым из базы данных, которое заключается в том, что последние должны быть целостны и непротиворечивы. Под целостностью данных подразумевается соответствие имеющейся в базе данных информации её внутренней логике, структуре и явно заданным правилам. Любое правило, направленное на ограничение возможного состояния базы данных,называют ограничением целостности. Помимо соблюдения целостности данных, также необходимо чтобы они были непротиворечивыми – это означает, что в базе данных нет логического противоречия.

В случае системы планирования производства, в качестве постоянных данных требуется хранить информацию о каждом запуске системы для того, чтобы можно было затем выбрать оптимальный план работы предприятия. Это ведет к тому, что появляется несколько версий одних и тех же данных (например названий технологических карт или разных версий одного заказа) и приводит к необходимости организации хранения и извлечения этих версий. Соответственно, разрабатываемая база данных должна быть доступна только на запись новых значений (не модификацию) и чтение, что позволит сохранить информацию о предыдущих запусках и добавлять новую.

Для организации хранилища, была выбрана реляционная модель базы данных. Данный выбор обусловлен тем, что данная модель получила широкое распространение, что является сильным аргументом как при внедрении системы (потому что реляционные баз данных организованны на многих предприятиях), так и при разработке, так как это основной тип баз данных при изучении таковых. Также реляционная модель обладает мощным математическим аппаратом, который основан на теории множеств, что позволяет анализировать, обосновывать и оптимизировать структуру разработанной базы данных и действия над ней.

3.2 Понятия реляционной теории

Для организации и теоретического обоснования описанного хранилища, использовалась теория реляционных баз данных.

Теория реляционных баз данных оперирует понятиями отношения (relation, от которого и пошло само название теории и баз данных основанных на ней), атрибута, домена и кортежа.

Атрибут – именованный столбец отношения. В пределах одного атрибута все значения должны быть одного типа данных, то есть принадлежать одному домену [11].

Домен – тип данных, множество всех допустимых значений атрибута [11].

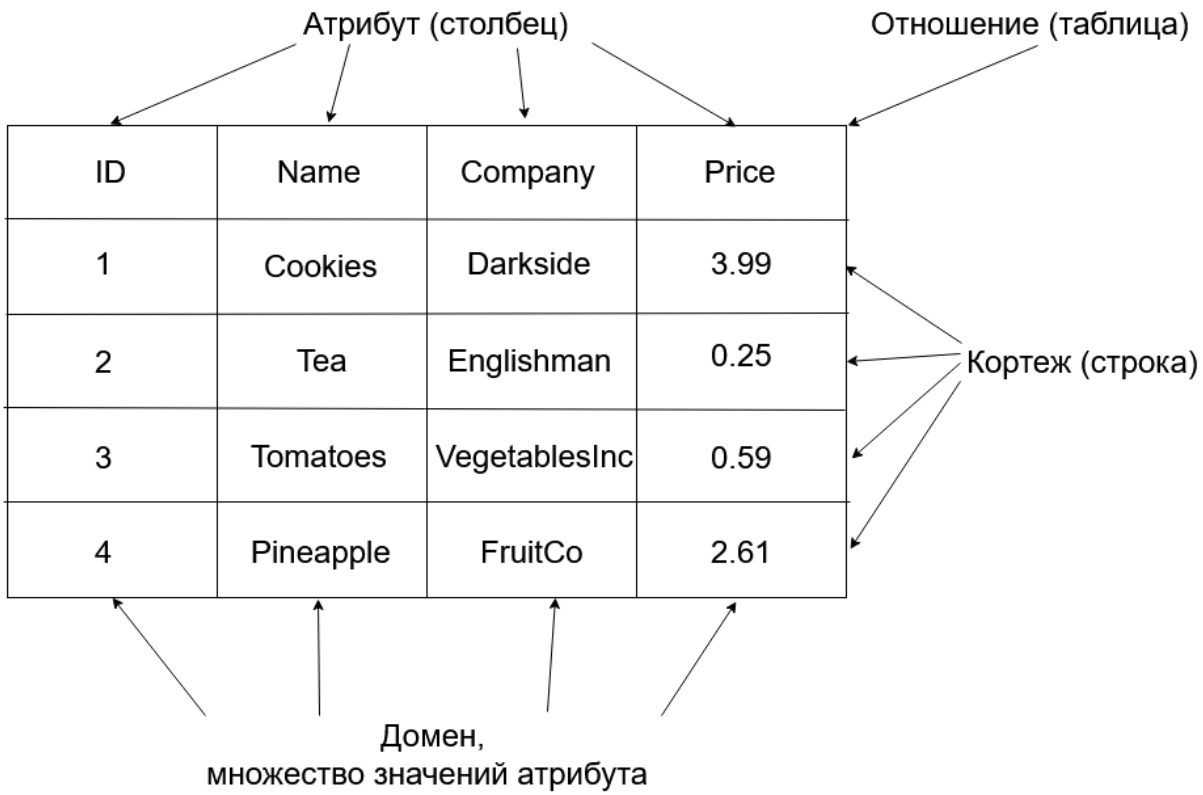

Рисунок 3.1 — Понятия реляционной базы данных

Кортеж – упорядоченный набор из N элементов, где N – это число атрибутов отношения [11]. Иначе говоря, кортеж – это строка или запись таблицы.

Отношение – множество упорядоченных N-кортежей [11]. Другими словами отношение – это двумерная (плоская) таблица, состоящая из столбцов и строк – атрибутов и кортежей (см. рисунок 3.1).

Также необходимо ввести понятие целостности по ссылкам заключающееся в отсутствии в любом её отношении внешних ключей (атрибут, указывающий на атрибут в другом отношении и совпадающий с ним по значению), ссылающихся на несуществующие кортежи [12]. $FK(R, a)$ – внешний ключ, ссылающийся на атрибут a на отношения R.

Так как любое отношение это множество, то им свойственны все операции определенные для множеств [13]:

— пересечение;

— объединение;

— вычитание;

— декартово произведение.

Помимо этих четырех операций, в теории реляционной алгебры, вводится еще 4 операции свойственные только отношениям [13]:

— выборка ($\sigma_\phi(R)$) – накладывает ограничение ϕ на отношение R;

— проекция ($\pi_\phi(R)$) – отображает только те атрибуты отношения R, которые были представлены в ϕ;

— соединение ($R_1 \bowtie_\phi R_2$) – соединяет кортежи из двух отношений R_1 и R_2 по условию ϕ;

— переименование ($\rho_{a,b}(R)$) – переименовывает атрибут отношения R из a в b .

Также в реляционной алгебре, для приближения к SQL (Structure Query Language, структурированный язык запросов), вводится оператор группировки ($_\phi\gamma$) – группировка кортежей по атрибутам заданным в ϕ.

Под записью данных в отношение будем понимать объединение исходного и нового отношений:

$$\forall R, R' : R \leftarrow R \cup R'$$

3.3 Формализация свойств базы данных

Для достижения согласованности данных необходимо чтобы были запрещены операции модификации и удаления данных, а запись новых кортежей в базу данных обладала двумя свойствами:

1) запись во все отношения позволяет получить любую предыдущую версию данных;

2) корректная запись в одно из отношений (при соблюдении соответствующих условий, о которых речь пойдет далее) не нарушает согласованности в базе данных.

Пусть заданы два отношения R_1 и R_2, такие что:

$$\begin{aligned} R_1 &= < name_1, version, ... > \\ R_2 &= < FK(R_1, name_1), version, ... > \end{aligned}$$

Для атрибута $version$ данных отношений задано требование монотонного возрастания:

$$\forall t_1 \in \mathbb{Z}, t_2 \in \mathbb{Z}, f : M \in \mathbb{Z} \to \mathbb{Z} : t_2 > t_1 \Rightarrow f(t_2) > f(t_1) \tag{3.1}$$

Например, в качестве самой простой и очевидной монотонно возрастающей функции можно взять функцию линей зависимости версии от времени $version(t) = t$, где t будет количеством секунд с начала эпохи Unix.

Определение 1. *Версионное соединение – это такое соединение двух отношений R_1 и R_2, для которых справедливо:*

$$\begin{aligned} &\forall R_1,\, R_2 \;:\; R_1.name_1 \;\supset\; R_2.name_1 : \\ &R_1 \overrightarrow{\bowtie} R_2 \;\Leftrightarrow\; {}_{R_2.version}\gamma(\sigma_{R_1.version \leqslant R_2.version}(R_1 \;\bowtie_{R_1.name_1 = R_2.name_1}\; R_2)) \end{aligned} \tag{3.2}$$

Также будем обозначать отношение $\forall n : R'_n$ как отношение которое полностью повторяющее набор атрибутов R_n.

Определение 2. *Объединение двух версионных соединений R_1 и R_2 осуществляется по следующему правилу:*

$$\overrightarrow{\cup} : \; R_1 \overrightarrow{\bowtie} R'_2 \to R_1 \overrightarrow{\bowtie} R'_2 \to R_1 \overrightarrow{\bowtie} R'_2$$

$$\begin{aligned} &\forall R_1, R_2, R'_1, R'_2 : R'_2.name \subset (R_1.name \cup R'_1.name), \\ &\qquad R_1.version \leqslant R'_2.version, R'_1.version > R_2.version : \\ &(R_1 \overrightarrow{\bowtie}\; R_2) \overrightarrow{\cup} (R'_1 \;\overrightarrow{\bowtie}\; R'_2) = (R_1 \cup R'_1) \;\overrightarrow{\bowtie}\; (R_2 \cup R'_2) \end{aligned} \tag{3.3}$$

Во всех случаях не описанных выше объединение не может быть произведено.

Обозначим версионное соединение отношений R_1 и R_2, как $\overrightarrow{J}$:

$$R_1 \overrightarrow{\bowtie} R_2 \Leftrightarrow \overrightarrow{J}$$
$$R'_1 \overrightarrow{\bowtie} R'_2 \Leftrightarrow \overrightarrow{J}'$$

Теорема 1. *Любая выборка из версионного соединения $\overrightarrow{J}$ входит в выборку из объединения $\overrightarrow{J}$ и любого допустимого версионного соединения $\overrightarrow{J}'$ (отношения R_1 и R'_1 и R_2 и R'_2 одинаковы по набору атрибутов) по условиям $\overrightarrow{\cup}$.*

$$\forall \overrightarrow{J}, \overrightarrow{J}' : \sigma_\phi(\overrightarrow{J}) \subset \sigma_\phi(\overrightarrow{J} \overrightarrow{\cup} \overrightarrow{J}') \tag{3.4}$$

Доказательство. Если предположить, что теорема 1 неверна, тогда объединение $\overrightarrow{J}$ и $\overrightarrow{J}'$ не включают в себя $\overrightarrow{J}$ для любого условия ϕ. Предположим что $\phi = true$, тогда выборка из $\overrightarrow{J}$ будет содержать полное множество отношения $\overrightarrow{J}$:

$$\forall \overrightarrow{J} : \overrightarrow{J} \Leftrightarrow \sigma_{true}(\overrightarrow{J}) \Rightarrow \overrightarrow{J} \not\subset \overrightarrow{J} \overrightarrow{\cup} \overrightarrow{J}'$$

По определению $\overrightarrow{J} \Leftrightarrow R_1 \overrightarrow{\bowtie} R_2$:

$$R_1 \overrightarrow{\bowtie} R_2 \not\subset (R_1 \overrightarrow{\bowtie} R_2) \overrightarrow{\cup} (R'_1 \overrightarrow{\bowtie} R'_2) \tag{3.5}$$

По определению 2:

$$R_1 \overrightarrow{\bowtie} R_2 \not\subset (R_1 \cup R'_1) \overrightarrow{\bowtie} (R_2 \cup R'_2) \tag{3.6}$$

По определению объединения множеств $R_1 \cup R_2 = \{x \mid x \in R_1 \mid x \in R_2\}$, что говорит о том, что результирующее множество содержит полное множество R_1 и R_2, следовательно :

$$\forall R'_1, R'_2 : \begin{array}{l} R_1 \not\subset (R_1 \cup R'_1) \\ R_2 \not\subset (R_2 \cup R'_2) \end{array} \tag{3.7}$$

Выражения 3.7 опровергают определение объединения множеств, что приводит к выводу о неверности предположения. Из этого следует, что теорема 1 верна, что и требовалось доказать. □

Пусть заданы три отношения R_1, R_2 и R_3, такие что:

$$\begin{aligned} R_1 &= < name_1, version_1, ... > \\ R_2 &= < FK(R_1, name_1), name_2, version_2, ... > \\ R_3 &= < FK(R_2, name_2), version_3, ... > \end{aligned}$$

Теорема 2. *Для любого версионного объединения* $(R_2 \overrightarrow{\bowtie} R_3) \overrightarrow{\cup} (R'_2 \overrightarrow{\bowtie} R'_3)$ *существует корректное версионное соединение получившегося отношения с* R_1.

$$R_1 \overrightarrow{\not\bowtie} (R_2 \overrightarrow{\bowtie} R_3) \Rightarrow R_1 \overrightarrow{\not\bowtie} ((R_2 \overrightarrow{\bowtie} R_3) \overrightarrow{\cup} (R'_2 \overrightarrow{\bowtie} R'_3)) \tag{3.8}$$

Доказательство. Пусть не существует верное соединение при верном версионном объединении:

$$R_1 \overrightarrow{\not\bowtie} (R_2 \overrightarrow{\bowtie} R_3) \overrightarrow{\cup} (R'_2 \overrightarrow{\bowtie} R'_3)) \tag{3.9}$$

Следовательно не выполняется одно либо оба условия существования версионного соединения:

1) $((R_2 \overrightarrow{\bowtie} R_3) \overrightarrow{\cup} (R'_2 \overrightarrow{\bowtie} R'_3)).name_1 \not\subset R_1.name_1$;

2) $((R_2 \overrightarrow{\bowtie} R_3) \overrightarrow{\cup} (R'_2 \overrightarrow{\bowtie} R'_3)).version_3 < R_1.version_1$

Если рассматривать первый вариант, то необходимо, используя определение 2, упростить выражение 3.9:

$$R_1 \overrightarrow{\not\bowtie} ((R_2 \cup R'_2) \overrightarrow{\bowtie} (R_3 \cup R'_3)) \tag{3.10}$$

Тогда, используя определение отношений:

$$(R_2 \cup R'_2).name_1 \not\subset R_1.name_1 \tag{3.11}$$

Так как в начальный момент времени система находилась в состоянии согласованности, то $R_2.name_1 \subset R_1.name_1 \Rightarrow R'_2 \not\subset R_1.name_1$. Данное утверждение является нарушением целостности по ссылкам, так что любая система управления базами данных запретит объединение $R_2 \cup R'_2$.

Рассматривая второе условие:

$$((R_2 \cup R'_2) \overrightarrow{\bowtie} (R_3 \cup R'_3)).version_3 < R_1.version_1 \tag{3.12}$$

Тогда, используя определение отношений:

$$(R_3 \cup R'_3).version_3 < R_1.version_1 \tag{3.13}$$

Так как в начальный момент времени система находилась в состоянии согласованности, то:

$$R_1.version_1 \leqslant R_3.version3 \Rightarrow \begin{cases} R'_3.version_3 < R_1.version_1 \\ R'_3.version_3 < R_3.version_3 \end{cases} \tag{3.14}$$

Из данной системы следует нарушение монотонности возрастания значения версий (новая версия меньше старой), что является нарушением условий версионного объединения.

В результате, оба варианта получения несогласованного состояния в базе данных ограничиваются либо системой управления базами данных, либо заданными определениями, что доказывает теорему 2.

Так как теорема 2 справедлива для 3 отношений, можно сказать что данная теорема будет справедлива и для $\forall N \in \mathbb{Z}$. □

Пусть есть N отношений таких, что:

$$\begin{aligned} R_1 &= < name_1, version_1, ... > \\ R_2 &= < FK(R_1, name_1), name_2, version_2, ... > \\ R_3 &= < FK(R_2, name_2), name_3, version_3, ... > \\ &... \\ R_N &= < FK(R_{N-1}, name_{N-1}), version_N, ... > \end{aligned}$$

Тогда, из теоремы 2 следует, что первое условие существования версионного объединения никак не меняется и полностью зависит от R_2, которое в свою очередь каскадно зависит от последующих отношений. Второе условие, по причине того, что в результирующем версионном объединении в качестве версии будет выбрано самое последнее значение, то есть $R_N.version_N$, соответственно если взять $N = 3$, то получится уже доказанная теорема, из чего следует, что теорема справедлива для $\forall N \in \mathbb{Z}$.

3.4 Структура базы данных

В разрабатываемой базе данных необходимо хранить:

— данные о заказах;

— данные о типах продукции;

— данные о продукции, которая относится к каждому заказу;

— данные о самой последовательности операций и самих операциях для каждого типа продукции;

— данные о каждой модели ресурсов;

— данные о связях между моделью ресурса и операциями.

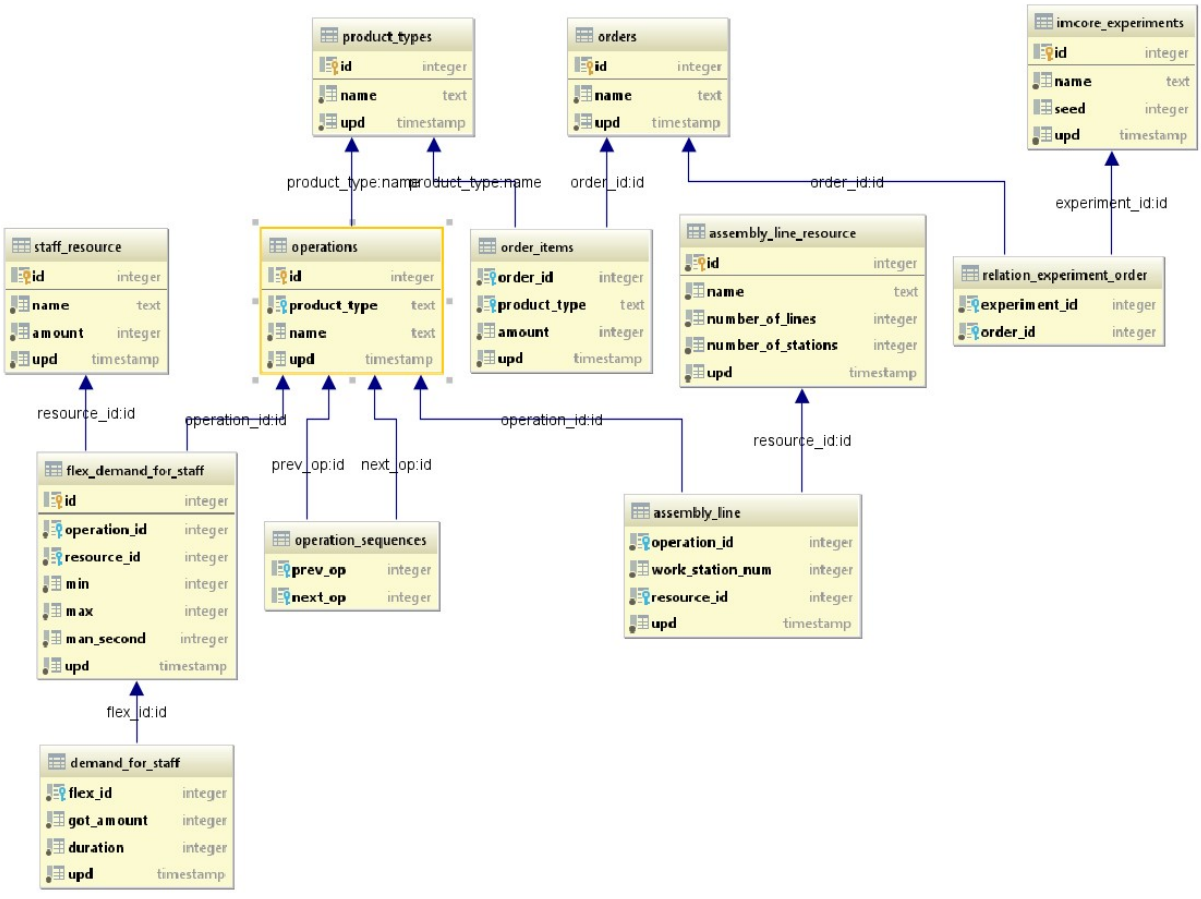

Рисунок 3.2 — Схема базы данных

Помимо этого, нужно учитывать, что кроме простого хранения данных, необходимо соблюдать “версионность”, ввиду того, что карта технологического процесса может меняться во времени. Значит и в

базе данных требуется следить за тем, чтобы в любой момент времени было возможно использовать любую из версий данной карты, иначе новый расчет оперативного плана (при условии, что другие данные остались неизменными) приведет к созданию новой версии плана, а, следовательно, предыдущую версию восстановить будет либо очень сложно, либо, в худшем случае, невозможно.

Для этого, учитывая определенные в разделе 3.3 теоремы и операции, все отношения были разделены две категории:

— внешний ключ является ссылкой на поле уникального идентификатора (id);

— внешний ключ является ссылкой на текстовое поле имени (name), а также имеет поле “version”.

Эти типы впоследствии были связаны между собой и позволили создать более гибкую структуру базы данных С помощью этой метки можно получить как последние данные из базы, просто максимизируя метку “version”, так и данные на определенный момент времени ограничивая эту метку необходимым временем.

Заключение

В рамках данной работы были рассмотрены, разработаны и протестированы компоненты системы планирования производства, а также организованно хранилище данных.

По итогу выполнения работы были достигнуты следующие результаты:

— произведен анализ отображения логического на физическое время, синтезирован, реализован и протестирован алгоритм отображения логического времени на физическое;

— проанализирована сборочная линия (её функции и ограничения, накладываемые на перемещение продукции), реализована и протестирована модель сборочной линии, впоследствии интегрированная в подсистему имитационного моделирования СПП;

— проанализированы данные предприятия, которые необходимо хранить в базе данных, синтезирована и обоснована её структура, проведено доказательство основных положений структуры на основе реляционной теории.

Разработанные модули в настоящий момент интегрированы в СПП с соответствующими интеграционными тестами.

Структура базы данных была реализована и используется для хранения и извлечения тестовых данных, что используется для тестирования подсистемы имитационного моделирования. Также были реализованны введенные в процессе обоснования структуры базы данных операторы и выражения, что упростило работу с версионированными отношениями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Roth T. P., Song Y., Burns M. J. et al. Cyber-Physical systems development environment for energy applications. 2017. URL: https://www.nist.gov/publications/cyber-physical-system-development...

2. Олег Новиков. Что такое индустрия 4.0? Цифры и факты. 2015. URL: http://holzex.ru/chto-takoe-industriya-4-0-tsifryi-i-faktyi.

3. Интеллектуальные технологии цифрового производства: Tech. Rep.: / Кремлев А.С., Маргун А.А., Юрьева Р.А. [и др.]: 2018.

4. Э. Мюллер, М. Шенк, З. Вирт. Планирование и эксплуатация промышленных предприятий: Рабочие методики для адаптивного, сетевого и ресурсосберегающего предприятия. 2017.

5. Дж. Лодон, К. Лодон. Управление информационными системами. 2005.

6. Д. О'Лири. ERP системы. Современное планирование и управление ресурсами предприятия. Выбор, внедрение и эксплуатация. 2004.

7. Гибсон, Дж.Л. Организации: поведение, структура, процессы. 2000.

8. Маззулло Джим, Уитли Питер. SAP R/3 для каждого. Пошаговые инструкции, практические рекомендации, советы и подсказки. 2008.

9. Герхард Келлер, Томас Дикерсбах Йорг. Планирование и управление производством с помощью решений SAP ERP. 2011.

10. Система планирования и прогнозирования. Пояснительная записка к техническому проекту: Tech. Rep.: / Кремлев А.С., Маргун А.А., А.А. Иващенко [и др.]: 2018.

11. Serge Abiteboul Richard Hull Victor Vianu. Foundations of Databases. 1994.

12. П Грей. Логика, алгебра и базы данных. 1989.

13. В.М. Илюшечкин. Основы использования и проектирования баз данных. 2008.

ПРИЛОЖЕНИЕ А

Листинг 1 содержит реализацию привязки операции к модели сборочной линии.

Листинг 1 — Метод привязки операции к модели

```
1   func (assembly *AssemblyLine) Bind(conf interface{}, events
        ...*imcore.ConcreteEvent) {
2       assembly.checkConfig(conf)
3       if assembly.remain == nil {
4           assembly.remain =
                make(map[*imcore.ConcreteEvent][]*imcore.ConcreteEvent)
5           assembly.complete =
                make(map[*imcore.ConcreteEvent][]*imcore.ConcreteEvent)
6       }
7       for _, e := range events {
8           assembly.remain[e] = events
9       }
10      assembly.initEvent(conf, events...)
11  }
```

Листинг 2 содержит реализацию метода проверки возможности выполнения операции моделью сборочной линии.

Листинг 2 — Метод ограничения

```
1   func (assembly *AssemblyLine) Constrain(event *imcore.ConcreteEvent)
        (int64, bool) {
2       lineNum, position, onLine := assembly.findProduct(event.ProductSN,
            event.ProductType)
3       if onLine {
4           if assembly.lineReady(lineNum) {
5               return assembly.lines[lineNum].maxTime, true
6           } else if !assembly.containsEvent(position, event) {
7               return assembly.lines[lineNum].maxTime, false
8           }
9       } else {
10          num, canLoad := assembly.canLoad()
11          if !canLoad {
12              return assembly.lines[0].minTime, false
13          }
14          lineNum = num
15      }
16      return assembly.lines[lineNum].minTime, true
17  }
```

Листинг 3 содержит реализацию завершения операции моделью сборочной линии.

Листинг 3 — Метод выполняющий завершение операции

```
1    func (assembly *AssemblyLine) Done(event *imcore.ConcreteEvent) {
2
3        lineNum, _, onLine := assembly.findProduct(event.ProductSN,
             event.ProductType)
4        if event.EventType == imcore.BeginEvent {
5            if !onLine {
6
7                assembly.loadProduct(event)
8            } else {
9                if assembly.lineReady(lineNum) {
10                   assembly.moveLine(lineNum, event)
11               }
12           }
13       } else {
14           assembly.finishOperation(event)
15       }
16       assembly.assertBind(event)
17       assembly.complete[event] = assembly.remain[event]
18       delete(assembly.remain, event)
19   }
```

Листинг 4 содержит реализацию загрузки продукта на одну из линий модели.

Листинг 4 — Метод загрузки продукта на одну из линий

```
1    func (assembly *AssemblyLine) loadProduct(event *imcore.ConcreteEvent)
         bool {
2        _, position, onLine := assembly.findProduct(event.ProductSN,
             event.ProductType)
3        if !onLine && position != -1 {
4            minLine, canLoad := assembly.canLoad()
5            if canLoad {
6                line := assembly.lines[minLine]
7                if !line.stations[0].isWorking && len(assembly.queue) > 0 {
8
9                    station := line.stations[0]
10                   if len(assembly.remainEvents[0][makeProduct(event)]) >
                         0 {
11                       station.isWorking = true
12                   }
13                   station.time = line.minTime
```

```
14                      station.product = assembly.queue[position]
15                      station.isEmpty = false
16                      line.stations[0] = station
17                      assembly.lines[minLine] = line
18
19                      queue := assembly.queue
20                      if position < len(queue)-1 {
21                          queue = append(queue[:position],
                                queue[position+1:]...)
22                      } else {
23                          queue = queue[:position]
24                      }
25                      assembly.queue = queue
26                      log.Printf("Line (%d) %s loaded on station 0",
                            minLine, station.product.String())
27                      return true
28                  }
29              }
30          }
31          return false
32      }
```

Листинг 5 содержит реализацию сдвига линии модели.

Листинг 5 — Метод сдвига линии

```
1       func (assembly *AssemblyLine) moveLine(lineNum int, event
            *imcore.ConcreteEvent) bool {
2           if !assembly.lineReady(lineNum) {
3               return false
4           }
5           line := assembly.lines[lineNum]
6           line.minTime = line.maxTime
7
8           if !line.stations[len(line.stations)-1].isEmpty {
9               log.Printf("Line (%d) %s unloaded from line (timestamp: %d)",
                    lineNum,
                    line.stations[len(line.stations)-1].product.String(),
                    line.minTime)
10          }
11          for i := len(line.stations) - 1; i > 0; i-- {
12              station := line.stations[i]
13              prevStation := line.stations[i-1]
14              station.product = prevStation.product
15              station.isEmpty = prevStation.isEmpty
16              station.time = line.minTime
17              if !station.isEmpty {
```

```
18                  station.isWorking = true
19              }
20              prevStation.product = Product{
21                  SerialNumber: -1,
22                  TypeName:     "",
23              }
24              prevStation.isEmpty = true
25              prevStation.isWorking = false
26              line.stations[i] = station
27              line.stations[i-1] = prevStation
28              if !station.isEmpty {
29                  log.Printf("Line (%d) %s changing station (%d -> %d)",
                        lineNum, station.product.String(), i-1, i)
30              }
31          }
32          assembly.lines[lineNum] = line
33          return true
34      }
```

Листинг 6 содержит реализацию окончания операции к модели сборочной линии.

Листинг 6 — Метод окончания операции

```
1       func (assembly *AssemblyLine) finishOperation(event
            *imcore.ConcreteEvent) {
2           lineNum, position, onLine := assembly.findProduct(event.ProductSN,
                event.ProductType)
3           if onLine {
4               station := assembly.lines[lineNum].stations[position]
5               product := makeProduct(event)
6               triggers := assembly.remainEvents[position][product]
7               for i, trigger := range triggers {
8                   if trigger == event {
9                       if i+1 < len(triggers) {
10                          triggers = append(triggers[:i], triggers[i+1:]...)
11                      } else {
12                          triggers = triggers[:i]
13                      }
14
15                      log.Printf("Line (%d) %s changing timestamp (%d)",
                            lineNum, station.product.String(), *event.Value)
16                      station.time = *event.Value
17                      if len(triggers) < 1 {
18                          station.isWorking = false
19                      }
20
```

```
21                      if assembly.lines[lineNum].maxTime < station.time {
22                          assembly.lines[lineNum].maxTime = station.time
23                      }
24                      assembly.lines[lineNum].stations[position] = station
25                      assembly.remainEvents[position][product] = triggers
26                  }
27              }
28
29              if !station.isWorking {
30                  assembly.moveLine(lineNum, event)
31              }
32          } else {
33              panic("Not queued product done!")
34          }
35      }
```

Листинг 7 содержит реализацию определения на какой линии находится продукт дял данной операции.

Листинг 7 — Метод, производящий определение линии по заданной продукции

```
1   func (assembly *AssemblyLine) findProduct(productSN imcore.ProductSN,
        productType imcore.ProductTypeName) (int, int, bool) {
2       for lineNum, line := range assembly.lines {
3           for stationNum, station := range line.stations {
4               if station.product.SerialNumber == productSN &&
                    station.product.TypeName == productType {
5                   return lineNum, stationNum, true
6               }
7           }
8       }
9
10      for pos, item := range assembly.queue {
11          if item.SerialNumber == productSN && item.TypeName ==
                productType {
12              return -1, pos, false
13          }
14      }
15
16      return -1, -1, false
17  }
```

Листинг 8 содержит реализацию проверки возможности загрузки продукта на одну из линий модели.

Листинг 8 — Метод, определяющий возможность загрузки продукции на одну из линий

```
1   func (assembly *AssemblyLine) canLoad() (int, bool) {
2       if len(assembly.queue) > 0 && len(assembly.lines) > 0 {
3           lineWithMinWorkTime := -1
4           assemblyMinWorkTime := int64(0)
5           for i, line := range assembly.lines {
6               if !line.stations[0].isWorking && line.stations[0].isEmpty
                    {
7                   if (assemblyMinWorkTime > line.minTime) ||
                        (lineWithMinWorkTime == -1) {
8                       assemblyMinWorkTime = line.minTime
9                       lineWithMinWorkTime = i
10                  }
11              }
12          }
13          if lineWithMinWorkTime != -1 {
14              return lineWithMinWorkTime, true
15          }
16      }
17      return -1, false
18  }
```

Листинг 9 содержит реализацию проверки готовности линии к сдвигу.

Листинг 9 — Метод, производящий проверку готовности линии к сдвигу

```
1   func (assembly *AssemblyLine) lineReady(lineNum int) bool {
2       for i, station := range assembly.lines[lineNum].stations {
3           product := Product{station.product.TypeName,
                station.product.SerialNumber}
4           if station.isWorking && !station.isEmpty &&
                len(assembly.remainEvents[i][product]) > 0 {
5               return false
6           }
7       }
8       station := assembly.lines[lineNum].stations[0]
9       if !station.isWorking && len(assembly.queue) > 0 &&
            station.isEmpty {
10          return false
11      }
12      return true
13  }
```

ПРИЛОЖЕНИЕ В

Листинг 10 содержит функцию для запуска отображения логического времени на физическое. Также определяет в зависимости от знака поданного логического времени какой расчет будет запущен.

Листинг 10 — Публичная функция для запуска извне

```
1  func LogicalToPhysTime(physStarts int64, logicalTime int64, config
       ConverterConfig) (int64, int64) {
2      confNum, wasPrinted := checkConfig(config)
3
4      if !wasPrinted {
5          log.Printf("New config %d: %+v\n", confNum, config)
6      } else {
7          log.Printf("Found config %d\n", confNum)
8      }
9      log.Printf("Start time is: %v (%v)", physStarts,
           time.Unix(physStarts, 0).UTC().Format("02/01/2006 15:04:05"))
10     log.Printf("Given logical time is: %v (%v in hours)", logicalTime,
           logicalTime/secondsInHour)
11
12     if logicalTime == 0 {
13         first, _ := mapLogicalToPhysTime(physStarts, logicalTime,
               config, true)
14         _, second := mapLogicalToPhysTime(physStarts, logicalTime,
               config, false)
15         return first, second
16     } else if logicalTime > 0 {
17         return mapLogicalToPhysTime(physStarts, logicalTime, config,
               false)
18     }
19     second, first := mapLogicalToPhysTime(physStarts, -logicalTime,
           config, true)
20     return first, second
21 }
```

Листинг 11 содержит основной алгоритм отображения. Возвращает два значения: начало и конец интервала, на который отображается логическое время.

Листинг 11 — Алгоритм отображения

```
1  func mapLogicalToPhysTime(physStarts int64, logicalTime int64, config
       ConverterConfig, isReverted bool) (int64, int64) {
2      curOffset := int64(0)
3      curTime := physStarts
```

```
4           first := int64(0)
5           isLeftFound := false
6           log.Println(" Logical time | Physical time")
7           log.Println("———————————————|———————————————————————————————")
8           for curOffset <= logicalTime {
9               curDay := timeToDate(curTime, isReverted)
10              intervals := config.getSchedule(curTime, isReverted)
11
12              if len(intervals) < 1 {
13                  curTime = nextDay(curTime, isReverted)
14                  continue
15              }
16
17              for _, interval := range intervals {
18                  log.Printf(" %-13v| %v (%v)", curOffset, curTime,
                        time.Unix(curTime, 0).UTC().Format("02/01/2006
                        15:04:05"))
19                  curOffset, curTime = interval.appendIntervalDur(curTime,
                        curOffset, curDay, isReverted)
20                  if curOffset >= logicalTime {
21                      curTime = appendTime(curTime, logicalTime-curOffset,
                            isReverted)
22
23                      if !isLeftFound {
24                          first = curTime
25                          curOffset = logicalTime
26                          isLeftFound = true
27                          break
28                      } else {
29                          break
30                      }
31                  }
32              }
33          }
34          log.Println("Result:")
35          log.Println("———————————————————————————————————————")
36          log.Printf(" %-13v| [%-19v - %-19v]", logicalTime, first, curTime)
37          log.Printf(" %-13v| [%-19v - %-19v]", logicalTime,
                time.Unix(first, 0).UTC().Format("02/01/2006 15:04:05"),
                time.Unix(curTime, 0).UTC().Format("02/01/2006 15:04:05"))
38          return first, curTime
39      }
```

Листинг 12 содержит метод реализующий прибавление или вычитание интервала в зависимости от направления расчета.

Листинг 12 — Добавление рабочего интервала

```
func (interval WorkInterval) appendIntervalDur(physTime int64,
    logicalTime int64, day int64, isReverted bool) (int64, int64) {
    if isReverted {
        if (interval.Ends + day) >= physTime {
            logicalTime += physTime - day - interval.Starts
        } else {
            logicalTime += interval.Ends - interval.Starts
        }
        physTime = day + interval.Starts
    } else {
        if (interval.Starts + day) <= physTime {
            logicalTime += interval.Ends - physTime + day
        } else {
            logicalTime += interval.Ends - interval.Starts
        }
        physTime = day + interval.Ends
    }
    return logicalTime, physTime
}
```

Листинг 13 содержит реализацию метода получения смен занятых сотрудников. В зависимости от направления расчета может “развернуть” список интервалов.

Листинг 13 — Метод получения расписания смен занятых сотрудников

```
func (config ConverterConfig) getSchedule(physTime int64, isReverted
    bool) []WorkInterval {
    currentDay := timeToDate(physTime, isReverted)

    scheduleName := config.getTemplateName(currentDay)
    intervals :=
        config.ScheduleForDay.getIntervals(physTime-currentDay,
        scheduleName, isReverted)
    scheduleName = config.getTemplateName(currentDay - oneDayInSecs)
    prevIntervals :=
        config.ScheduleForDay.getIntervals((physTime-currentDay)+oneDayInSecs,
        scheduleName, isReverted)

    if len(prevIntervals) > 0 {
        for i, interval := range prevIntervals {
            if interval.Starts-oneDayInSecs >= 0 ||
                interval.Ends-oneDayInSecs > 0 {
                prevIntervals[i].Starts = interval.Starts -
                    oneDayInSecs
```

```
13                      prevIntervals[i].Ends = interval.Ends - oneDayInSecs
14                      intervals = append([]WorkInterval{prevIntervals[i]},
                            intervals...)
15                  }
16              }
17          }
18          if isReverted && len(intervals) > 1 {
19              last := len(intervals) - 1
20              for i := 0; i <= last/2; i++ {
21                  intervals[i], intervals[last-i] = intervals[last-i],
                        intervals[i]
22              }
23          }
24
25          return intervals
26      }
```

Листинг 14 содержит реализацию извлечения смен из конфигурации по заданному шаблону рабочего дня.

Листинг 14 — Извлечение смен из конфигурации по заданному шаблону

```
1       func (schedules ScheduleForDay) getIntervals(physTime int64,
            scheduleName DayTemplateName, isReverted bool) []WorkInterval {
2           result := make([]WorkInterval, 0)
3           intervals := schedules[scheduleName]
4
5           for _, interval := range intervals {
6               if interval.contains(physTime, isReverted) {
7                   result = append(result, interval)
8               }
9           }
10          return result
11      }
```

Листинг 15 содержит реализацию получения названия шаблона рабочего дня из даты.

Листинг 15 — Получение названия шаблона рабочего дня из даты

```
1       func (config ConverterConfig) getTemplateName(date int64)
            DayTemplateName {
2           exception, ok := config.ExceptionDates[date]
3           if ok {
4               return exception
5           }
6           if exception != "" {
```

```
7               return exception
8           }
9           for item := range config.ScheduleTemplate {
10              if item == int(time.Unix(date, 0).UTC().Weekday()) {
11                  return config.ScheduleTemplate[item]
12              }
13          }
14          panic("No schedule defined")
15      }
```