МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»

ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

Корниенко Мария Алексеевна

РАЗРАБОТКА ПРОГРАММНОЙ НАДСТРОЙКИ ДЛЯ ОБЪЕДИНЕНИЯ ТЕХНОЛОГИЧЕСКИХ ОПЕРАЦИЙ

Выпускная квалификационная работа (уровень бакалавриата)

Направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. В. Высочин  
Доцент, к. т. н. (подпись, дата)

К ЗАЩИТЕ ДОПУСКАЮ:

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Милюков  
Профессор, д. ф.-м. н. (подпись, дата)

Симферополь, 2022

# Реферат

Корниенко М. А. Разработка программной надстройки для объединения технологических операций // Выпускная квалификационная работа (уровень бакалавра) по специальности 09.03.04 Программная инженерия / Кафедра компьютерной инженерии и моделирования Физико-технического института Крымского федерального университета им. В. И. Вернадского. – Симферополь, 2022. – 86 с., 34 рис., 11 ист.

*Объект разработки* – программная надстройка MES-системы Zenith SPPS для объединения технологических операций.

*Цель работы* – разработка программной надстройки MES-системы Zenith SPPS на языке программирования Delphi (Object Pascal) для объединения однотипных технологических операций при наличии возможности такого объединения.

*Методы исследования и аппаратура* – изучение информационных источников, таких как официальный сайт системы Zenith SPPS с материалами информационного характера, «Руководство разработчика интерфейса» системы Zenith SPPS и др., изучение работы самой системы, анализ, функциональное тестирование; персональный компьютер с установленной демонстрационной версией системы Zenith SPPS и IDE Embarcadero Delphi 10.4.2 Sydney.

*Результат работы* – в ходе выполнения практической части выпускной квалификационной работы была написана программа, решающая задачу объединения технологических операций, и проведено ее функциональное тестирование.

РАЗРАБОТКА, DELPHI, MES-СИСТЕМА, APS-СИСТЕМА, ТЕХНОЛОГИЧЕСКИЙ ПРОЦЕСС, ТЕХНОЛОГИЧЕСКАЯ ОПЕРАЦИЯ, ПРОГРАММНАЯ НАДСТРОЙКА, БАЗЫ ДАННЫХ, СУБД, SQL, OLE AUTOMATION, OLE DB, ADO, COM.

# Оглавление

[Реферат 2](#_Toc106628004)

[Оглавление 3](#_Toc106628005)

[Список сокращений и условных обозначений 5](#_Toc106628006)

[Введение 6](#_Toc106628007)

[Глава 1 Основные понятия информационных систем планирования и управления производством 8](#_Toc106628008)

[1.1 Понятие информационной системы 8](#_Toc106628009)

[1.1.1 Основные этапы в развитии ИС 9](#_Toc106628010)

[1.2 Классификация ИС 9](#_Toc106628011)

[1.3 ERP-системы 12](#_Toc106628012)

[1.4 MES-системы 14](#_Toc106628013)

[1.4.1 Интегративный подход в использовании MES и ERP систем 15](#_Toc106628014)

[1.5 APS-системы 16](#_Toc106628015)

[1.6 APS/MES-система Zenith SPPS 17](#_Toc106628016)

[1.6.1 Особенности MES-системы Zenith SPPS 19](#_Toc106628017)

[1.6.2 Интеграционные возможности системы Zenith SPPS 20](#_Toc106628018)

[1.7 Технологический процесс и его элементы 21](#_Toc106628019)

[1.7.1 Понятие технологической операции 21](#_Toc106628020)

[1.7.2 Принципы построения технологического процесса 24](#_Toc106628021)

[1.7.3 Структура рабочего времени и норма штучного времени 24](#_Toc106628022)

[1.8 Задача объединения технологических операций 26](#_Toc106628023)

[1.9 Вывод по главе 27](#_Toc106628024)

[Глава 2 Проектирование программной надстройки 28](#_Toc106628025)

[2.1 Промежуточное ПО как способ взаимодействия программ 28](#_Toc106628026)

[2.1.1 Категории промежуточного ПО 28](#_Toc106628027)

[2.1.1.1 ПО для организации межпрограммного взаимодействия 28](#_Toc106628028)

[2.1.1.2 ПО для организации доступа к базам данных 30](#_Toc106628029)

[2.1.2 Microsoft Component Object Model 31](#_Toc106628030)

[2.1.3 Типы взаимодействия программ 32](#_Toc106628031)

[2.2 OLE-автоматизация 33](#_Toc106628032)

[2.3 Прямой доступ к данным OLE-сервера из стороннего приложения 34](#_Toc106628033)

[2.3.1 Навигационные методы 34](#_Toc106628034)

[2.3.2 Реляционный доступ к данным 35](#_Toc106628035)

[2.3.3 Технология ADO 36](#_Toc106628036)

[2.3.3.1 Компоненты Delphi для поддержки ADO 38](#_Toc106628037)

[2.4 Выбор программно-технической среды реализации 39](#_Toc106628038)

[2.4.1 Требования к аппаратному обеспечению и программно-технической среде 39](#_Toc106628039)

[2.5 Вывод по главе 40](#_Toc106628040)

[Глава 3 Реализация и тестирование программной надстройки для объединения технологических операций 41](#_Toc106628041)

[3.1 Подготовка и настройка программной среды реализации 41](#_Toc106628042)

[3.2 Создание объекта ZenApp и подключение к внутренней БД 43](#_Toc106628043)

[3.3 Реализация объединения технологических операций 44](#_Toc106628044)

[3.3.1 Получение атрибутов выбранной в Zenith SPPS операции 44](#_Toc106628045)

[3.3.2 Вывод возможных для объединения операций 45](#_Toc106628046)

[3.3.3 Осуществление проверок на возможность объединения 46](#_Toc106628047)

[3.3.4 Объединение операций 48](#_Toc106628048)

[3.3.5 Вид интерфейса для объединения операций 48](#_Toc106628049)

[3.4 Программная реализация задачи разделения технологических операций 49](#_Toc106628050)

[3.4.1 Вывод разъединяемых операций 49](#_Toc106628051)

[3.4.2 Разъединение операций 50](#_Toc106628052)

[3.4.3 Вид интерфейса разъединения операций 51](#_Toc106628053)

[3.5 Функциональное тестирование программной надстройки 52](#_Toc106628054)

[3.6 Вывод по главе 60](#_Toc106628055)

[Заключение 61](#_Toc106628056)

[Список литературы 62](#_Toc106628057)

[Приложение 1 Листинг разработанной программной надстройки для объединения технологических операций на языке программирования Delphi 63](#_Toc106628058)

# Список сокращений и условных обозначений

АО Аппаратное обеспечение

БД База данных

ИС Информационная система

ОС Операционная система

ПО Программное обеспечение

СУБД Система управления базами данных

ТП Технологический процесс

ADO ActiveX Data Object – Объект данных ActiveX

APS Advanced Planning & Scheduling – Усовершенствованное планирование

COM Component Object Model – Объектная модель компонентов

DLL Dynamically Loadable Library – Динамически загружаемая библиотека

MES Manufacturing Execution System – Система управления производством

OLE Object Linking and Embedding – Присоединенные и встроенные объекты

SQL Structured Query Language – Язык структурированных запросов

# Введение

Системы управления и планирования производства являются одной из наиболее важных и значимых частей современного производственного процесса, и при этом благодаря широкому перечню своих возможностей их организация зачастую очень сложна. В связи с этим, возникает необходимость упрощения процесса работы в них, чтобы еще больше повысить показатели эффективности и производительности. В частности, для этого можно каким-то образом укрупнять операции – т.е. объединять задачи, которые пользователям необходимо выполнять в соответствии с их должностными обязанностями. Например, задача объединения технологических операций при помощи программной надстройки возникла, потому что присоединение однотипных операций друг к другу по одной неудобно и занимает больше времени, чем выбор сразу всего необходимого множества и его объединение, а также по причине неэффективности регистрации и администрирования отчетности на каждую такую небольшую операцию вместо того, чтобы администрировать все подобные операции совместно при наличии такой возможности.

Эту проблему невозможно решить на этапе расчета расписания или до того, поскольку неизвестно заранее, как именно алгоритмы оптимизации MES-системы расположат на производственном графике те или иные операции. Нерешенность данной проблемы приводит к тому, что время сотрудников и ресурсы оборудования расходуются неэффективно, а сам процесс эксплуатации MES-системы Zenith SPPS не является удобным для пользователей.

Целью данной работы являлась разработка программной надстройки MES-системы Zenith SPPS на языке программирования Delphi (Object Pascal) для объединения однотипных технологических операций при наличии возможности такого объединения.

Для достижения поставленной цели необходимо решить следующий ряд задач:

1. Изучить особенности работы APS/MES-системы Zenith SPPS, непосредственно относящиеся к вышеобозначенной проблеме.

2. Составить алгоритм объединения технологических операций, учитывающий все тонкости и нюансы.

3. Продумать подходящий для решения задачи и максимально интуитивно понятный (user-friendly) и удобный интерфейс.

4. Реализовать программную надстройку, реализующую объединение и разъединение операций, в соответствии с п.2 и п.3 данного списка в виде динамически загружаемой библиотеки на языке Delphi с целью интеграции ее как дополнительного модуля в APS/MES-систему Zenith SPPS.

5. Провести функциональное тестирование полученного решения.

Решение данной частной задачи позволит улучшить систему Zenith SPPS, а новые знания, полученные в процессе достижения поставленной цели, впоследствии позволят проектировать более усовершенствованные и производительные APS/MES-системы.

# Глава 1 Основные понятия информационных систем планирования и управления производством

## 1.1 Понятие информационной системы

Информационная система (ИС) – система, занимающаяся различными видами работы с информацией, а также всевозможные организационные ресурсы, которые обеспечивают эту работу. В 149-ФЗ «Об информации, информационных технологиях и о защите информации» от 27.07.2006 информационная система определяется как «совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств». Современный взгляд на ИС предусматривает, в основном, использование компьютеров как главного технического средства работы с информацией, а также важность роли человека, без которого не было бы смысла в создании таких систем и через которого работа с информацией и происходит.

В своем жизненном цикле ИС получает, организует, хранит, трансформирует и выдает результирующие данные или производит изменение своего состояния. Основной продукт труда и предмет в ИС – это информация.

Основными свойствами, общими для всех ИС, являются то, что в основе каждой такой системы лежит среда хранения и доступа к данным, соответствующая предметной области ИС, а также простота, понятность и в то же время необходимый уровень функциональности графического интерфейса, ориентированного на конечного пользователя.

Во время своей работы ИС обслуживают два вида потоков информации – входящий (когда происходит ввод новой информации) и исходящий (при выдаче имеющейся информации по запросам). Очень важно, чтобы ответ на запросы пользователей выдавался системой быстро и в полной мере, поскольку обслуживание клиентов является наиболее приоритетной задачей. Это гарантируется наличием во внутреннем устройстве стандартных процедур поиска информации и хранением ее в определенном порядке.

В основе любой информационной системы лежит процесс переработки, хранения и доступа к данным.

ИС состоит из 5 компонентов:

1. Аппаратное обеспечение (АО) – помимо ПК, мобильных устройств и различных распределенных компьютерных систем, также устройства ввода/вывода информации, периферийное, телекоммуникационное оборудование и т.д.

2. Программное обеспечение (ПО) – системное (например, операционная система) и прикладное ПО.

3. Телекоммуникационные (сетевые) технологии – проводные и беспроводные сетевые технологии.

4. Базы данных (БД) и хранилища данных (ХД).

5. Человеческие ресурсы – различный технический персонал, такой как бизнес-аналитики, системные аналитики, разработчики, системные архитекторы, специалисты по информационной безопасности, а также любые работники организации, работающие с ИС.

### 1.1.1 Основные этапы в развитии ИС

ИС впервые возникли в 50-х гг. прошлого столетия. Сперва они могли только заниматься обработкой счетов и расчетами заработной платы, т.к. реализовывались на электромеханических бухгалтерских счетных машинах. Тем ни менее, в некотором роде это сокращало временные затраты на подготовку нужных документов.

В 60-е гг. отношение к ИС изменилось, т.к. полученные из них данные, стали применяться для разнообразной периодической отчетности. Для таких задач организациям потребовалось более мощное многофункциональное компьютерное оборудование.

В 70-е - начале 80-х ИС все чаще и чаще используются как средство управленческого контроля, ускоряющее процесс принятия решений.

К концу 80-х гг. концепция использования ИС снова претерпевает изменения. Теперь они становятся стратегическим источником информации и используются в любых организациях на всех уровнях. Здесь ИС уже помогают организациям достичь успеха в своей деятельности, создавать новые товары и услуги, находить новые рынки сбыта, обеспечивать себе достойных партнеров, организовывать выпуск продукции по низкой цене и т.д.

## 1.2 Классификация ИС

Существует множество классификаций информационных систем по различным признакам. Например, в зависимости от масштаба ИС могут быть одиночными, групповыми и корпоративными. Одиночные ИС, как правило, располагаются на одном ПК, содержат несколько простых приложений, связанных между собой одной БД, из расчета на работу одного или нескольких пользователей на одном рабочем месте и создаются с помощью локальных СУБД (таких как Microsoft Access, dBase, Paradox и т.д.). Групповые ИС, как понятно из названия, больше подходят для коллективного (группового) пользования и обычно построены на базе ЛВС с использованием серверов БД (или, иначе, SQL-серверов – например, Oracle, Microsoft SQL Server, DB2 и др.). Корпоративные ИС используются в крупных компаниях и в связи с этим могут поддерживать территориально разнесенные узлы, чаще всего имеют иерархическую структуру и многоуровневую или клиент-серверную архитектуру. В них также используются сервера БД, как и в групповых информационных системах, но при этом гораздо выше требования к надежности и целостности данных, что обеспечивается поддержкой ссылок и транзакций.

Помимо этих трех групп в некоторых классификациях выделяют еще очень большие (Very Large) и сверхбольшие (Ultra Large) ИС, при этом определяя их вкупе с корпоративными информационными системами (КИС) как работающие в интранете (аналог Интернета, только внутри корпорации или учреждения) и Интернете.

В свою очередь, КИС (или их подсистемы) в зависимости от уровня обслуживания производственных процессов на предприятии могут быть разделены на классы:

1. Класс A: Системы управления технологическими объектами и/или процессами.

2. Класс B: Системы подготовки и учета производственной деятельности предприятия.

3. Класс C: Системы планирования и анализа производственной деятельности предприятия.

Изначально системы класса А появились по причине важности, проблематичности и трудоемкости учета различных материалов на предприятии (учет материалов является основной сферой применения таких систем).

Обычно системы класса А обладают следующим рядом свойств:

* высокая автоматизация реализуемых функций;
* ярко выраженный контроль за состоянием объекта управления;
* наличием механизма обратной связи (петли обратной связи);
* быстрой обработкой данных (имеется в виду время от получения информации о состоянии объекта управления до выдачи управляющего воздействия на него);
* слабой корреляцией между состояниями объектов управления и подсистемы/системы управления.

Примерами таких систем могут служить распределенные системы управления (DCS), автоматизированные системы управления технологическими процессами, системы последовательного управления и др.

Далее появились системы планирования производственных (или же материальных, в зависимости от направления работы предприятия) ресурсов, которые уже относятся к классу B. Данные системы вошли в стандарты MRP (Material Requirements Planning – планирование потребности в материалах) и MRP II (Manufacturing Requirements Planning – планирование производственных ресурсов). Они являются широко распространенными (в большей степени, на Западе), т.к. успешны в применении, особенно в производственных отраслях.

В основе систем стандартов MRP и MRP II лежат следующие принципы:

* производственная деятельность описывается в виде потока связанных между собой заказов;
* при выполнении заказов учитываются разного рода ограничения ресурсов;
* производственные циклы и запасы минимизируются;
* заказы производства (и снабжения) формируются на основе производственных графиков и заказов реализации.

Помимо этого, конечно, у данных стандартов есть и множество других функций, таких как, например, планирование загрузки рабочих мест, оборудования и планирование цикла технологической обработки. Но важно заметить, что системы стандартов MRP и MRP II решают скорее проблему управления материальными ресурсами организации, а не только проблему учета ресурсов.

К системам класса B можно отнести вышеупомянутые MRP и MRP II, а также MES (Manufacturing Execution Systems – системы управления производством), CRM (Customer Relationship Management – системы управления взаимоотношениями с клиентами), CAD (Computer-aided design – автоматизированные системы проектирования, САПР), CAM (Computer-aided manufacturing – автоматизированные системы поддержки производства), APS (Advanced Planning & Scheduling – усовершенствованное планирование), различные системы учета и т.д. Основной предпосылкой к созданию таких систем стала необходимость выделения на уровне технологических подразделений обособленных задач управления.

В область применения систем (подсистем) класса С входят следующие задачи:

* на основе информации, приходящей из систем класса B, анализировать деятельность предприятия;
* планировать деятельность организации;
* планировать и распределять ресурсы, находящиеся у нее в управлении;
* создавать и контролировать исполнение производственных заданий;
* взаимодействовать с персоналом (управляющим субъектом) при выполнении их задач;
* регулировать глобальные параметры работы предприятия;
* интерактивно обрабатывать информацию.

К системам класса С традиционно относят ERP (Enterprise Resource Planning – планирование ресурсов предприятия), EIS (Executive Information System – «исполнительная» система поддержки принятия решений), АСУП (автоматизированная система управления предприятием) и др.

Также можно классифицировать информационные системы по типам деятельности, которые ИС обслуживают. В этой классификации выделяют финансовые и учетные системы, производственные системы, системы маркетинга, кадровые системы и пр., выполняющие различные вспомогательные функции в соответствии с деятельностью фирмы.

## 1.3 ERP-системы

Из всех систем класса C наибольшую популярность обрели системы стандарта ERP (Enterprise Resource Planning – планирование ресурсов предприятия). Первые ERP-системы появились в конце 80-х годов прошлого столетия, а сама концепция ERP была предложена в 1990 году аналитиком Gartner Ли Уайли и явилась следующим шагом в развитии стандарта MRP II.

ERP-система – это ПО, предназначенное для управления и интеграции основных бизнес-процессов на предприятии (финансы, цепочки поставок, управление ресурсами, в том числе управление персоналом) в единую систему.

ERP-система состоит из модулей - программных компонентов, каждый из которых фокусируется на обособленном бизнес-процессе. Некоторые модули (такие как финансовый, кадровый (HR), отношения с клиентами (CRM), закупки) – считаются основными и развертываются первыми практически для каждого типа бизнеса.

ERP отличается от автономных приложений тем, что имеет свою централизованную БД, в которой находится вся информация по бизнес-транзакциям и других выполняемых в различных модулях действиях, а также механизмом интеграции, позволяющим модулям взаимодействовать между собой и с общей БД. Таким образом, все модули содержат актуальную информацию, и система имеет согласованный вид, в отличие от ПО для управления бизнесом, написанного разными вендорами. Схематическая модель работы ERP-системы приведена на рисунке 1.1. Также современные ERP-системы, в отличии от старых с монолитной архитектурой, которые приходилось долго и сложно дорабатывать под каждую отдельно взятую организацию, хорошо интегрируются с другими системами, что позволяет удовлетворить требования практически любой компании.

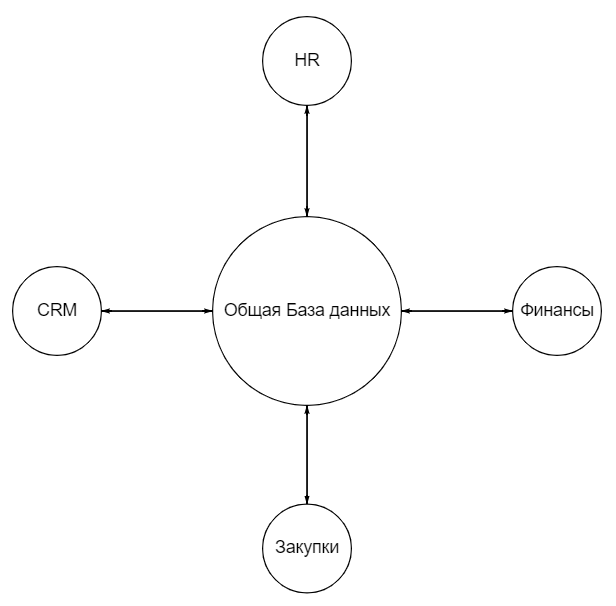


Рисунок 1.1. Упрощенная модель работы ERP-системы.

В последние десятилетия в Российской Федерации прогрессирует тенденция снижения производительности труда на производственных предприятиях. Это может быть вызвано недостаточной информированностью о текущем состоянии бизнес-процессов, о нагрузке обслуживающего персонала и самих рабочих, о стадиях жизненного цикла производимого товара, о производительности организации в целом – и эти недостатки как раз решаются внедрением ERP-систем, поскольку в них область управления предприятием помимо всего прочего расширяется на основе автоматизации процессов.

Помимо этого, что немаловажно, современные ERP-решения обеспечивают повышенную безопасность и конфиденциальность данных, настройку с минимальными элементами программирования, а также аналитику, позволяющую быстрее внедрять новые решения, что обеспечивает устойчивое развитие.

Термин ERP-система употребляется в двух значениях:

1. Как информационная система для планирования ресурсов предприятия;

2. Как методология эффективного планирования и управления ресурсами предприятия, которые необходимы для всех областей деятельности и бизнес-процессов организации.

В своей функциональности ERP-системы включают, помимо складского учета еще и управление материалами, учитывая все ресурсы предприятия, в первую очередь денежные. Таким образом, ERP-системы должны охватывать все непосредственно связанные с его деятельностью сферы предприятия; осуществлять основные управленческие и финансовые функции.

## 1.4 MES-системы

MES–система (Manufacturing Execution System) – система управления производственными процессами. Предназначением таких систем являются задачи анализа, оптимизации, координации и синхронизации изготовления товаров и изделий в рамках производства. Хотя MES-системы являются системами управления на уровне цеха, они также нередко используются в целом для управления производством на предприятии в связке с ERP-системами верхнего уровня.

MES-системы активно используются в производстве еще с 90-х гг. ХХ века.

В ERP-системах, которые по сути своей скорее являются информационными корпоративными системами, системами для управления предприятием в целом, чаще всего планирование работ и технологических операций ведется на основе устаревшего стандарта MRP II, при этом, как правило, не учитывая загрузку оборудования в настоящий момент и текущее состояние изделий. Вследствие этого оперативно планы корректировать не удается, и весь процесс производства строится на жесткой дисциплине внутри всех участвующих в процессе подразделений. С помощью ERP-систем, в большинстве случаев, нельзя построить детальный план со сроками по технологическим операциям и изделиям, который требуется для управления на уровне цехов – только перерасчет заданий в объемах. MES-системы же напротив, формируют требования не к предприятию в целом, а к той его детали, для которой выстраивают рабочий план. Это может быть как подразделение, так и участок или цех. Даже функция управления документами, предусматриваемая в MES-системах по регламенту, имеет отношение к цеховым документам, таким как заказ, наряд, отчетная ведомость и т.д.

Самыми важными MES-функциями являются диспетчеризация производственных процессов в цеху и детальное (оперативно-календарное) планирование, поскольку именно они делают MES-систему системой оперативного характера, ориентированной на оперативное управление цеховыми процессами и расчет расписания работы цехового оборудования. Конечно, помимо станков в цеху, MES-система может также составлять расписание для бригад рабочих, транспортных средств и т.д., поскольку ее возможности планирования намного мощнее, чем у других классов систем.

Получая объем работы, выдаваемый APS-системой в виде план-графика работы цеха, либо запланированный ERP, MES-система выстраивает более точные расписания для приборов и оперативно отслеживает процесс их выполнения, поскольку она ориентирована на выполнение плана как можно лучше по экономическим показателям цеха. Она оптимизирует по ряду критериев план более общего характера, полученный ею из другой системы, учитывая при этом все необходимые детали, и после оптимизации вследствие уплотнения расписания появляются свободные участки времени оборудования и возможности взять дополнительные заказы, что очень экономически выгодно.

Также, поскольку в MES-системах учитывается гораздо больше факторов и зачастую критериев построения расписания, они оперируют со значительно меньшими размерностями назначения, нежели APS-системы. Также исключительно MES-системы могут описаться на т.н. векторные (интегральные) критерии при построении расписания (когда 1 критерий состоит из нескольких частных). При этом диспетчер, составляя расписание, может указать, что он хочет видеть в конкретном расписании – уменьшение календарной длительности выполнения всего задания, уменьшение длительности операций переналадок, высвобождение станков, имеющих небольшую загрузку и т.п. Помимо этого, перерасчет расписания в MES-системе может происходить с интервалом до 1 минуты, что позволяет в режиме реального времени контролировать производственный процесс и заранее предугадать нарушения расписаний, чтобы иметь возможность вовремя на них отреагировать и даже исправить.

Важным свойством MES-системы является то, что ее расписание выполнимо, чего не могут гарантировать другие классы систем планирования производства, поскольку их алгоритмы очень гибкие и позволяют корректировать расписание при появлении любых отклонений в производственных процессах (на которые они реагируют очень чутко). Они являются незаменимыми в позаказном и мелкосерийном производстве.

Также в отличие от ERP и APS-систем, MES-системы более предметно ориентированы – на какую-то конкретную отрасль промышленности. Поэтому суть производственных процессов в них отражается максимально точно и полно.

### 1.4.1 Интегративный подход в использовании MES и ERP систем

В современной промышленности нет никакого конфликта между MES и ERP-системами, потому что в совокупности они привносят оперативную ясность, которой не могут добиться по отдельности.

В ERP наибольшее внимание уделяется созданию и управлению планами и графиками относительно производства, использования ресурсов, доставки, отгрузки, и также сбору информации о самом бизнесе. MES, в свою очередь, сосредоточены на управлении и оперативном мониторинге производственных операций и различной отчетности о деятельности подразделений в режиме реального времени.

При совместном использовании ERP и MES, создается интегрированная экосистема, предлагающая целостный взгляд на закупки, финансы, управление поставками, производственную логистику и т.д. Это повышает гибкость и обеспечивает большую надежность данных, улучшая прогнозируемость во всех областях

ERP-система говорит о том, какой продукт производить, а MES-система добавляет информацию о производственных площадках, чтобы производить этот продукт с наибольшей прибылью и наименьшими потерями.

## 1.5 APS-системы

APS (Advanced Planning & Scheduling Systems) – системы расширенного и усовершенствованного планирования. Они появились в середине 90-х годов прошлого века и являются продвинутым инструментом планирования процесса производства на предприятии.

APS-системы, наряду с ERP, MES, MRP II, также относятся к системам, отвечающим за планирование в производстве.

Поскольку методы планирования, используемые в концепции MRP II, не обеспечивали требуемой достоверности планирования и точности во времени, задачей APS-систем стало решение задач по автоматизации SCM (Supply Chain Management – управление цепочками поставок), реализуемое благодаря планированию всех работ во времени с учетом загрузки мощностей – как для предприятия, так и для его частей – подразделений, цехов и участков. Поэтому APS-системы являются расширением и усовершенствованием как MRP II, так и концепции MES-систем в частности.

Как правило, APS-системы получают данные из модулей ERP-систем, поэтому работают также в связке. Ранее их функционал считался частью MES-систем, но к настоящему времени выделен в отдельный вид ИС предприятий.

Можем выделить три наиболее важные группы модулей в APS-системах:

* SCE (Supply Chain Execution) – оперативное управление логистической цепью;
* SCC (Supply Chain Configuration) – стратегическое конфигурирование цепи поставок;
* SCP (Supply Chain Planning) – тактико-оперативное планирование цепи поставок.

APS-системы очень тесно сопряжены с понятиями синхронности, оперативности, оптимизации и точности.

Синхронность выражается в том, что расписания и графики составляются для всех подразделений организации с учетом сроков поставок партнеров, и выходят из общего расписания работы всей организации; также в том, что можно одновременно планировать ресурсы и материалы и строить расписания, принимая во внимание реальную загрузку оборудования на текущий момент.

Оперативность понимается как возможность в максимально короткие временные рамки определить срок выполнения того или иного заказа. Однако оперативность коррекции расписания в текущий момент времени и диспетчерского контроля не относится к функциям APS-систем, поскольку скорее всего не даст никаких значимых результатов, если только не будут постоянно меняться внешние условия либо портфель заказов.

Точность и оптимальность составляемых расписаний и графиков должны характеризовать абсолютно любую систему планирования. По сравнению с более ранними стандартами, APS-системы учитывают одновременно гораздо больше факторов, формируя расписания, - мощности предприятия, их текущую и плановую нагрузку, потребности материалов, переналадки и др.

В основе алгоритмов построения расписания в APS-системах лежит некоторый итерационный процесс получения допустимого расписания с учетом внесения новых ограничений (от наиболее важных к наименее важным) на каждой итерации. Иногда этот процесс делают еще проще – планируя поочередно все необходимое множество деталей. Затем сформированное расписание для дальнейшей оптимизации передают уже в работу MES-системы для создания фактической реализации.

Соответственно, основной задачей APS-систем является расчет допустимого расписания для больших размерностей в минимальные сроки в рамках имеющихся вычислительных ресурсов, с целью чего и происходит упрощение алгоритмов расчета и отбрасывание не значащих (в рамках поставленной задачи на первоначальных этапах) факторов. Поэтому число критериев, учитываемых при планировании, в таких системах сильно ограничено.

Как уже упоминалось выше, достаточно часто APS-системы интегрируются с MES-системами, которые больше подходят для частого перепланирования расписания в режиме реального времени (в отличии от APS-систем, где коррекция расписания связана с частотой появления заказов в системе, а обратная связь считается избыточной), хотя и для меньших размерностей (чаще всего в рамках одного цеха или участка), и которые дают лучшую точность при планировании, учитывая гораздо большее число факторов.

## 1.6 APS/MES-система Zenith SPPS

Система Zenith SPPS (Zenith Shopfloor Production Planning System – Система планирования на уровне производства «Зенит») является APS/MES-системой, то есть объединяет функции APS с широким инструментарием MES-систем, а также имеет встроенный универсальный генератор внутрицеховой отчетности Zenith Report. Данная система выполняет полный цикл работ по управлению и сопровождению производственной деятельности в режиме реального времени: планирование, отслеживание, оптимизация и документооборот, начиная от регистрации заказа в системе и до его выполнения (выпуска готового изделия).

Программный комплекс Zenith SPPS осуществляет комплексное управление производством, оперативно-календарное планирование (расчёт, коррекцию и поддержку производственных графиков), автоматизацию документооборота и диспетчерский контроль за движением и наличием комплектующих.

Разрабатываясь с 1996 года, на настоящий момент система является одной из самых продуманных и доступных MES-систем в отечественном производстве. Zenith SPPS может быть как интегрирована в состав ИС организации (импортировать данные, содержащиеся в других приложениях или системах, таких как Microsoft Excel или 1С), так и работать автономно. Архитектурные особенности системы позволяют осуществлять эффективное взаимодействие практически с любыми программными продуктами других производителей, поскольку для обмена данными и доступа к информации используются общепринятые технологии (ADO, OLE Automation, BDE, WinSocket, Windows API и др.). Это доступно продемонстрировано на рисунке 1.2.

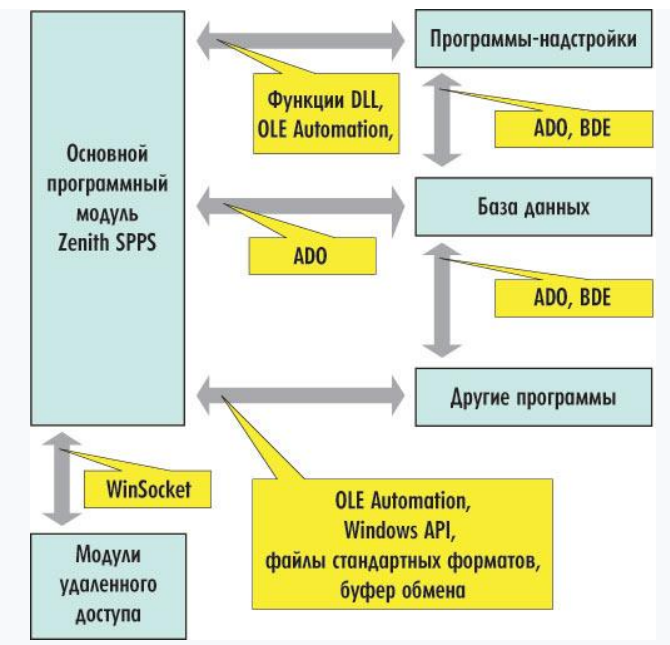


Рисунок 1.2. Взаимодействие с другими программами и доступ к данным в системе Zenith SPPS.

Ключевой особенностью системы является удачное сочетание широко используемых функций ERP бизнес-приложений с механизмами производственной диспетчеризации. Интерфейс приложения спроектирован для максимальной доступности и удобства конечного пользователя, а наличие подробной справочной документации обеспечивает эффективную работу с системой и минимальные временные затраты на изучение ее особенностей.

Также система Zenith SPPS обладает высокой адаптивностью к различным видам производства и процессам за счет универсальности своего ядра (например, ее можно использовать в таких отраслях как гостиничный бизнес, машиностроение, мебельное производство, электронная промышленность, строительство, металлургия и даже образовательный процесс).

### 1.6.1 Особенности MES-системы Zenith SPPS

Разработка системы Zenith SPPS изначально велась в соответствии с концепцией RAD (Rapid Application Development – быстрая разработка приложений), с использованием теории построения человеко-ориентированных пользовательских интерфейсов. Новые версии программы производятся примерно раз в полугодие в постоянном двустороннем процессе коммуникации с пользователями. Все нововведения и изменения в системе подробно фиксируются в сопроводительной документации.

MES-система Zenith SPPS занимается планированием ресурсов и поддержкой производственного расписания с учетом данных о загрузке оборудования и его состоянии, наличия свободных ресурсов в режиме реального времени.

Результирующие данные можно выгружать в другие бизнес-приложения в разнообразных форматах (XML, HTML, XLS и т.д.).

Перечислим основные отличительные особенности системы Zenith SPPS:

1. Универсальное высокоэффективное ядро, позволяющее быстро рассчитывать расписание из большого числа позиций, с выбором необходимых критериев для более персонализированной настройки алгоритма расчета. Система содержит различные инструменты для поддержки жизненного цикла, осуществляющие добавление новых заказов, коррекцию уже введенных в систему данных и удаление устаревших. Также есть возможность добавления новых данных и изменения графика работы без перерасчета всего расписания.

2. Удобный пользовательский интерфейс, ориентированный на психологические особенности потенциальных пользователей, поскольку при разработке системы использовалась теория построения человеко-ориентированных пользовательских интерфейсов. Помимо этого, MES-система Zenith SPPS эффективно использует стандартные возможности интерфейса ОС Microsoft Windows, будучи глубоко в нее интегрированной.

3. Работа в сети – система содержит автоматизированные рабочие места для различных групп персонала (диспетчеров цеха, мастеров производственных участков и технологов); каждая группа осуществляет свой ряд технологических процессов и операций соответственно с должностными обязанностями.

4. Гибкость и открытость – широкая адаптивность MES-системы Zenith SPPS для различных видов производства и сфер деятельности достигается благодаря универсальности системного ядра и многофункциональному пользовательскому интерфейсу. Система может функционировать как самостоятельно, так и совместно со сторонними программами, получая из них данные и выгружая свои.

5. Доступная стоимость, полученная благодаря использованию технологии RAD (стоимость лицензии и внедрения значительно ниже, чем у многих зарубежных аналогов). Система в базовой комплектации имеет возможность быстрой и беспроблемной установки.

Применение MES-системы Zenith SPPS значительно влияет на улучшение показателей производительности труда на предприятии, увеличение коэффициента загрузки оборудования, повышение процента соблюдения сроков поставки готовой продукции; система помогает поднять уровень управляемости производственных процессов, что положительно влияет на предприятие в целом.

Zenith SPPS используется на ряде предприятий в Российской Федерации.

### 1.6.2 Интеграционные возможности системы Zenith SPPS

Zenith SPPS может функционировать в качестве сервера приложений, поскольку данные программы хранятся в формате реляционной системы управления базами данных (СУБД). Это позволяет использовать такие средства, как язык SQL (Structured Query Language – структурированный язык запросов), пользовательские DLL (Dynamic Link Library – библиотеки динамической компоновки) и EXE-файлы, а также OLE Automation (Object Linking and Embedding – присоединенные и встроенные объекты, иногда еще называется OLE DB).

Система имеет широкие интеграционные возможности. В частности, можно разрабатывать плагины – программные надстройки, расширяющие функционал программы, и подключать их к основной программе в формате DLL. К настоящему времени в системе реализован ряд таких надстроек: надстройка для клонирования операций, надстройка «Планируемое время», надстройка для упрощенного расчета трудоемкости операций, для разбиения заказа на партии, надстройка «Вывод операций» для исключения некоторых технологических операций из производственного расписания.

Метод подсоединения к программе описан в небольшой инструкции, находящейся в сопроводительной документации «Руководство разработчика интерфейса». Там же перечислен ряд методов и свойств OLE, с помощью которых можно получить различные данные о запущенном приложении. Но для полноценной работы с программой разработчику программной надстройки необходим прямой доступ к данным – поскольку функции OLE предназначены скорее для программно-сложных вычислений и не рассчитаны на решение задач прямого доступа к данным.

Простая обработка данных осуществляется с помощью непосредственного обращения к встроенной БД (установление параллельного соединения к БД одновременно с самой системой Zenith SPPS). При этом в плане считывания данных из базы не возникнет никаких проблем, но при непосредственной записи данных в базу внесенные изменения отобразятся в основной программе не сразу. Данная особенность определяется ядром базы данных и работой драйвера (т.е. повлиять на это для разработчика программной надстройки нет никакой возможности).

## 1.7 Технологический процесс и его элементы

Первым исторически зафиксированным технологическим процессом стало описание процесса приготовления пива на глиняной табличке в древнем Шумере. С тех пор технологические процессы становились все более сложными и многоэтапными. В наши дни один технологический процесс может содержать в себе сотни и тысячи операций (шагов), иметь множество веток в соответствии с разнообразными условиями. Помимо выбора оборудования, инструментов и материалов, современный технологический процесс должен выполнять требования финансовых и плановых показателей, технических условий и др.

Согласно ГОСТу, технологический процесс (ТП) – это «часть производственного процесса, содержащая целенаправленные действия по изменению и (или) определению состояния предмета труда». Иными словами, это определенная последовательность выполнения расположенных в четко определенном порядке операций, с момента начала обработки заготовки до получения готового изделия. ТП является наиболее значимой частью производственного процесса, в который также входят административное управление, управление финансами, закупка, логистика, а также контроль качества.

Технологический процесс обязательно документируется в маршрутной, операционной и технологической картах, которые содержат его описание на различных уровнях и в разной степени детальности в соответствии с правилами оформления, установленными в Единой системе технологической документации (ЕСТД).

ТП состоит из некоторого количества технологических операций, а также установок, переходов и проходов.

### 1.7.1 Понятие технологической операции

Технологическая операция является завершенной частью технологического процесса, как правило, выполняется на одном рабочем месте без перерывов, до момента перехода к следующей детали. Технологическая операция является элементарным ТП, наименьшей его частью, которая обладает всеми его свойствами.

Технологические операции в системе Zenith SPPS изображаются на Графике загрузки рабочих мест в виде прямоугольников разных цветов, как показано на рисунке 1.3.

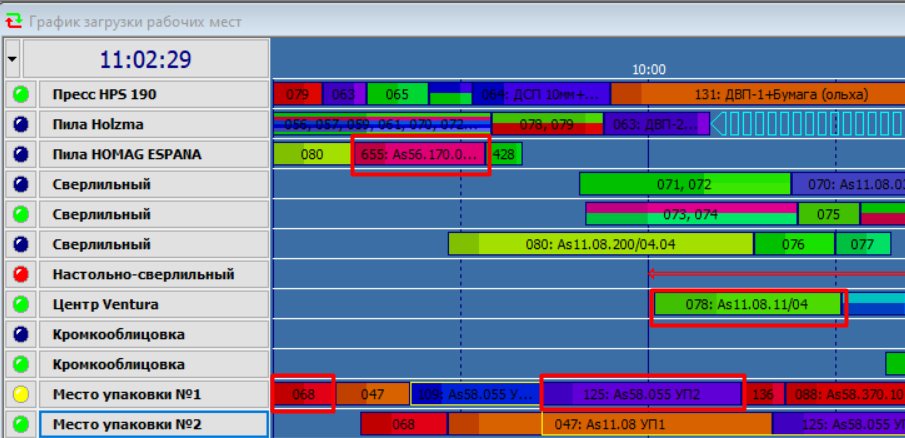


Рисунок 1.3. Изображение технологических операций в MES-системе Zenith SPPS.

Операции на графике связаны между собой зависимостями, следующая по технологическому процессу зависит от предыдущей, и в системе есть возможность детальной настройки зависимостей, в которой можно выбрать связь с предыдущей операцией (финиш-старт или старт-старт), а также варианты минимальной задержки (начало операции не ранее фиксированной даты, с опережением или задержкой, в определенный день недели и др.). Вид окна с настройкой зависимости продемонстрирован на рисунке 1.4.

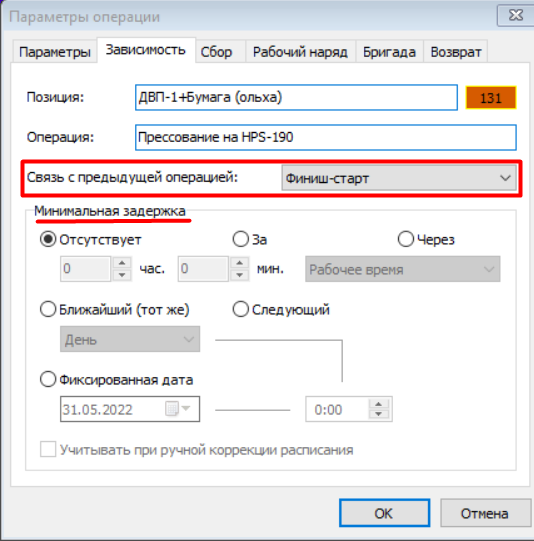


Рисунок 1.4. Окно настройки зависимости операции в MES-системе Zenith SPPS.

Как правило, одна отдельно взятая технологическая операция выполняется одним сотрудником на одном рабочем месте. В качестве примера можно привести распечатку отчета, прессование, упаковку, SQL-запрос к БД и др.

Также существует такое понятие, как установка – это некоторая часть операции, которая выполняется при условии того, что закрепление обрабатываемой детали не изменяется.

Технологические операции состоят из технологических переходов.

Переход (который, в свою очередь, состоит из проходов) – это часть технологической операции, во время которой обрабатываемая поверхность детали, инструмент и настройки оборудования остаются неизменными. Если несколько поверхностей одной детали в одно и то же время обрабатываются несколькими инструментами, считается, что это также происходит в рамках одного технологического перехода.

Проход – это часть технологического перехода, во время которой инструмент совершает одно перемещение в направлении подачи (например, снятие 1 слоя металла осуществляется за 1 проход).

Также можно выделить и другие элементы технологической операции, такие как рабочий и вспомогательный ход, позиция, наладка, подналадка, но здесь перечислены основные.

Технологические операции также разделяются на виды (названия операций могут иметь различающиеся названия в зависимости от специфики производства):

* операционные;
* маршрутные;
* маршрутно-операционные.

### 1.7.2 Принципы построения технологического процесса

Существует два ключевых принципа построения ТП:

1. Принцип укрупнения операций – в этом случае в рамках одной технологической операции собирается большое число переходов; таким образом, как правило, уменьшается общее количество установок, а также обеспечивается более точное взаимное расположение обрабатываемых поверхностей (поскольку они обрабатываются за одну установку). Данный принцип наиболее важен при изготовлении крупногабаритных и сложных деталей, т.к. их установка требует значительного количества времени. Согласно этому принципу осуществляют работу автоматы и полуавтоматы, токарные станки и т.д.

2. Принцип расчленения операций – принцип, эффективный при несложной конфигурации деталей и больших сериях, поскольку заключается в том, чтобы разбить процесс на простейшие операции, когда каждый переход выполняется за отдельную установку. Настройка оборудования в этом случае происходит лишь единожды (для первой детали), остальные детали в партии идут следом и обрабатываются согласно выполненной наладке. Количество установок значительно увеличивается, поэтому важно обеспечить быстроту и точность установок для каждого вида операций. Этот принцип делает организацию рабочего места более рациональной и автоматизированной.

### 1.7.3 Структура рабочего времени и норма штучного времени

Производительность труда работника на производстве зависит от количества изделий, обработку которых он производит в единицу времени (как правило, смена или час), находясь на определенном рабочем месте.

Рабочее время – это время, которое работник обязан проводить на рабочем месте за выполнением своих должностных обязанностей по своему трудовому договору и должностной инструкции. Рабочее время складывается из перерывов и собственно времени работы.

В свою очередь, время работы состоит из:

1. Подготовительно-заключительного времени – время подготовки работником рабочего места к выполнению заданной операции, а также действия, сопряженные с ее окончанием. Отличительной особенностью подготовительно-заключительного времени является то, что оно не зависит от объема операции, поэтому при объединении однотипных операций и, соответственно, увеличении суммарной длительности выполнения объединенной операции данный показатель будет оставаться таким же.

2. Оперативного времени – время выполнения данной работы непосредственно. Бывает основным или технологическим (время выполнения основной работы – самим рабочим или под его наблюдением) и вспомогательным (время, затрачиваемое на обеспечение выполнения основной работы, а также передвижения и т.д.; повторяется при выполнении определенного объема работ).

3. Времени обслуживания рабочего места (уход за рабочим местом и поддержание его в работоспособном состоянии).

В дальнейшем в рамках вычисления фактического времени выполнения технологической операции важны будут только первые две составляющие из этого списка.

Техническая норма времени – это штучное время Tшт, которое требуется для выполнения определенной технологической операции.

Норму времени можно рассчитывать относительно любого объема работы; если техническая норма времени устанавливается на единичный нормированный объем работы при выполнении технологической операции, то она называется нормой штучного времени.

Взаимосвязи норм времени отражены на рисунке 1.5.

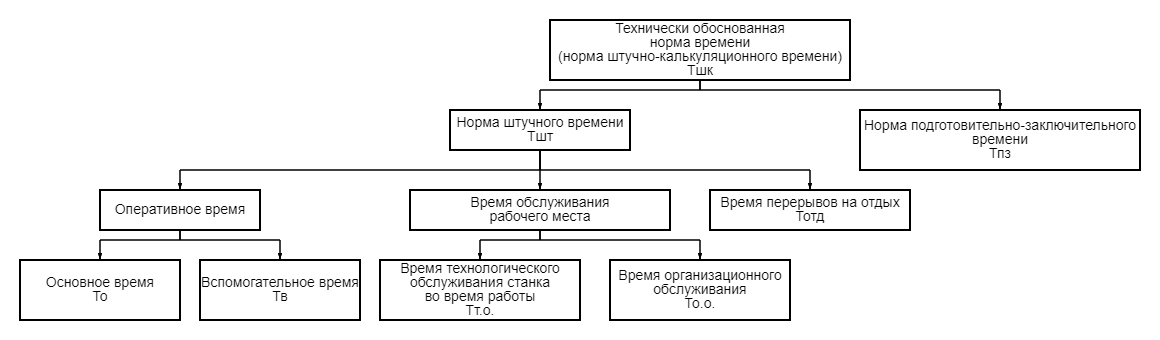


Рисунок 1.5. Виды и взаимосвязи норм времени.

Все виды затрат рабочего времени определяются на принятую единицу работы (операцию или штуку), и входят в норму штучно-калькуляционного времени (Тшк). Их соотношение математически отображается следующим образом:

(1.1)

(1.2)

где Тшк – это штучно-калькуляционное или полное время выполнения операции,

Тшт – штучное время,

Тпз – время подготовительно-заключительное,

То – основное (технологическое/машинное) время,

Тв – вспомогательное время,

То+ Тв – оперативное время,

Тт.о. – время технологического обслуживания станка во время работы,

То.о. – время организационного обслуживания (подготовка станка, уборка, передача сменщику),

Тотд – время на отдых и естественные потребности,

n – количество деталей в партии.

Существует два основных метода расчета технически обоснованных норм времени: аналитический метод (расчет по нормативам) и расчетно-сравнительный метод (с помощью изучения затрат рабочего времени наблюдением и расчета норм времени по типовым нормам).

## 1.8 Задача объединения технологических операций

В процессе эксплуатации системы стала заметна проблема, когда однотипные задачи решались отдельными операциями, иными словами множество мелких однотипных операций располагались друг за другом на одном рабочем месте, и на каждую такую операцию был необходим свой пакет документации, каждую нужно было регистрировать и администрировать, что неэффективно расходовало время сотрудников и ресурсы оборудования и делало более неудобным пользование MES-системой.

В связи с этим было предложено сделать следующее: спроектировать программную надстройку для объединения (и разделения при необходимости) операций, расположенных рядом по графику, в соответствии с принципами построения технологического процесса. Надстройка должна быть оформлена в виде DLL (Dynamically Loadable Library – динамически загружаемой библиотеки), для того чтобы была возможность интегрировать ее в MES-систему в виде дополнительного функционального модуля.

Данный процесс сопряжен с рядом сложностей: например, потери информации (о продолжительности каждой отдельной операции или уникальных идентификаторов операций), связанные с объединением ряда однотипных операций в одну более длительную, и ее восстановление при разъединении, потому как хранение дополнительных данных неудобно в связи с большим размером целевой таблицы. Помимо этого, одним из требований к разработке являлось сохранение целостности структуры внутренней базы данных (не удалять никакие записи и не добавлять новые, не перестраивать структуру таблиц, и пр.), поскольку разрабатываемая надстройка должна стать дополнительным функционалом к существующей системе.

Чтобы решить проблему, необходимо разработать программные средства, которые будут объединять идущие друг за другом операции:

1. Которые являются однородными, т.е. относятся к одному и тому же виду работ.

2. С учетом их технологических зависимостей, поскольку технологическое расписание нарушаться не должно.

Также следует рассмотреть случаи, когда в объединении участвуют уже объединенные операции или операции, находящиеся в сборе (либо, наоборот, нужно отсоединить одну или несколько операций от ранее объединенных). Помимо этого, нужно продумать удобный и подходящий под все перечисленные задачи интерфейс, который будет также интуитивно понятен для пользователя.

## 1.9 Вывод по главе

Существует множество различных по организации информационных систем, каждая из которых выполняет свои задачи в определенных сферах, решает свои проблемы. Однако не существует ни одной универсальной ИС.

APS и MES-системы относятся к классу систем планирования и анализа производственной деятельности предприятия, который, в свою очередь, является частью корпоративных информационных систем. Zenith SPPS – это APS/MES-система отечественной разработки.

Системы планирования и управления производством занимаются формированием и настройкой технологического процесса, состоящего из технологических операций, определенным образом зависящих друг от друга.

В данной выпускной квалификационной работе необходимо решить проблему, обнаруженную при использовании системы планирования и управления производством Zenith SPPS – проблему объединения (и разъединения ранее объединенных) технологических операций, которая на настоящий момент в системе не решена и влечет за собой неэффективное расходование времени персонала и ресурсов оборудования.

# Глава 2 Проектирование программной надстройки

## 2.1 Промежуточное ПО как способ взаимодействия программ

Промежуточное ПО (Middleware, программное обеспечение промежуточного слоя) – это класс программ, которые являются связующим звеном между операционной системой и работающими в ней приложениями. Промежуточное ПО обеспечивает управление данными в распределенных приложениях. Именно с помощью промежуточного программного обеспечения пользователи посредством запросов могут отправлять формы в браузере или возвращать серверу веб-страницы на основе своего пользовательского профиля.

К промежуточному ПО можно отнести:

* мониторы транзакций;
* программное обеспечение промежуточного слоя БД;
* ПО промежуточного слоя для взаимодействия в Интернете;
* промежуточное ПО сервера приложений и др.

Промежуточное программное обеспечение может объединять множество приложений в единую ИС.

ПО промежуточного слоя посредством общих API (Application Programming Interface – программный интерфейс приложения) позволяет установить сообщение между различными программами (в частности, функционирующими на разных платформах), которое не будут затрагивать вносимые в дальнейшем изменения в структуре этих приложений (только если эти изменения не будут касаться самого API промежуточного слоя).

### 2.1.1 Категории промежуточного ПО

Широкая популярность промежуточного программного обеспечения во внедрении и разработке комплексных систем повлекла за собой возникновение множества видов такого ПО, с разной степенью надежности, масштабируемости и принципами взаимодействия. Однако условно все эти типы по специализации можно разделить на две большие категории:

1. ПО, предназначенное для организации межпрограммного взаимодействия.

2. ПО для организации доступа к БД.

#### 2.1.1.1 ПО для организации межпрограммного взаимодействия

Программное обеспечение для организации межпрограммного взаимодействия создано с целью упрощения IPC (Inter-Process Communication – межпроцессное взаимодействие) или, иначе, IAC (Inter-Application Communication – межпрограммное взаимодействие), а также для облегчения распределенного доступа к объектам в инфраструктуре КИС. Основные технологии, использующиеся в таких программах:

1. RPC (Remote Procedure Call – удаленный вызов процедур): в этом случае, приложение может не только выполнять свои функции, процедуры и методы, но и процедуры, расположенные на удаленном сервере. Для этого оба процесса (клиентский и серверный) используют специальные объекты – «заглушки» (client stub и server stub – клиентская и серверная «заглушки» соответственно), не реализующие никакой логики и предназначенные исключительно для взаимодействия приложений. Все обращения клиента в виде названий функций и соответствующих им списков параметров собираются клиентской «заглушкой» в сетевые сообщения и передаются серверной «заглушке» (в технической литературе этот процесс называют marshalling – маршалинг, упорядочивание). Затем происходит распаковка данных серверной «заглушкой» (unmarshalling или демаршалинг в русскоязычной литературе), передача их в реальную серверную функцию, и упаковка результирующих данных в обратном порядке. Клиентская «заглушка» дожидается ответа, распаковывает полученные данные и возвращает их в исходную программу. Описание правил взаимодействия между клиентом и сервером в модели RPC происходит на языке IDL (Interface Description Language или Interface Definition Language – язык описания интерфейсов). С помощью IDL происходит изоляция RPC от языков программирования, на которых разработаны те или иные приложения – поскольку при обращении клиентом к удаленной процедуре, клиент применяет свои языковые конструкции, а компилятор IDL трансформирует их в понятные для сервера. Интерфейс IDL устроен по принципу черного ящика – содержит только имена функций, описание их параметров и результата выполнения.

2. MOM (Message-Oriented Middleware – связующее ПО, ориентированное на обработку сообщений): в таких системах взаимодействие клиента и сервера происходит посредством сообщений (управляющих команд и передаваемых данных в виде текстовых блоков). Передача сообщений идет через HTTP, POP/SMTP и другие байт-ориентированные протоколы, а обмен сообщениями происходит через API MOM-систем. Запросы сервисов помещаются в очередь сообщений, и их обработка происходит в порядке приоритетов, основываясь также на доступности ресурсов. Также возможна отложенная доставка (при появлении адресата сообщения в сети или же осуществляемая по расписанию). Сервер сообщает успешный либо неуспешный ответ о попытке выполнить затребованные операции. Помимо основанных на очередях, существуют также сервисы MOM, осуществляющие непосредственную передачу сообщений или передачу сообщений на основе подписки.

3. TPM (Transaction Processing monitors – монитор обработки транзакций): этот тип промежуточного ПО осуществляет контроль передачи данных серверу от клиента в распределенных БД. С его помощью сохраняется целостность данных, поскольку он постоянно производит проверки на наличие потерянных или незавершившихся транзакций. Транзакция – это один или несколько операций обращения к базе данных, представляющих собой законченную логическую единицу работы с данными. Для транзакций обязательно выполнение четырех принципов ACID (Atomicity – атомарность, Consistency – согласованность, Isolation – изоляция, Durability - прочность или долговременность). TPM обеспечивает выполнение перечисленных условий, а также концентрирует и перенаправляет запросы к распределенным базам данных.

4. DOS (Distributed object systems – распределенные объектные системы): здесь взаимодействие происходит между несколькими программными объектами. У каждого такого объекта есть свой уникальный сетевой идентификатор и свои публичные свойства и методы, посредством которых осуществляется доступ к предоставляемому им функционалу. Это похоже на рассмотренную выше концепцию RPC, даже также используется язык IDL, но, в отличии от RPC, распределенные объекты могут компоноваться во время работы приложения (динамически), а не лишь на этапе компиляции.

Наиболее популярными являются следующие архитектурные стандарты DOS:

* Microsoft COM (Component Object Model – объектная модель компонентов) – технологический стандарт от Microsoft для организации взаимодействия приложений в семействе ОС Windows. К этому стандарту относятся также COM+, DCOM (Distributed COM – расширенная версия COM для связи объектов в сетях), DirectX, Microsoft OLE Automation и др;
* EJB (Enterprise JavaBeans) – стандарт от Sun Microsystems, являющийся частью Java EE, для поддержки содержащих бизнес-логику серверных компонентов. Он значительно упрощает проектирование распределенных систем на Java;
* CORBA (Common Object Request Broker Architecture – типовая архитектура опосредованных запросов к объектам) – технология, созданная рабочей группой OMG (Object Management Group) для организации взаимодействия программ, запущенных на различных платформах и оборудовании, а также написанных на разных языках программирования. ORB (Object Request Broker – брокеры объектных запросов), разработанные по этому стандарту, создают интерфейсы для разработки клиент-серверных объектно-ориентированных систем.

#### 2.1.1.2 ПО для организации доступа к базам данных

Промежуточное ПО для работы с БД делится на 2 типа:

1. Собственное промежуточное программное обеспечение СУБД – встроенные механизмы доступа, отличные для разных серверов БД. К такому ПО относится поддержка стандартов SQL (определяется разработчиками для каждой конкретной СУБД самостоятельно), а также встроенные средства импорта/экспорта данных СУБД из внешних источников.

2. Основное промежуточное программное обеспечение баз данных – внешние интерфейсы (относительно самих баз данных), с помощью которых можно обращаться к базам данных. К ним относятся такие протоколы как, например, JDBC (Java DataBase Connectivity – стандарт соединения с различными СУБД на Java, включенный в состав Java SE) и ODBC (Open DataBase Connectivity – программный интерфейс для доступа к БД, аналог JDBC от Microsoft), а также концептуальные решения, как DQB (Distributed Query Broker - брокер распределенных запросов, децентрализованный стандарт доступа к данным в базе с помощью ORB) или EDA (Event-Driven Architecture – архитектура, управляемая событиями, и в отличие от DQB централизованная; шаблон управления КИС в зависимости от событий, происходящих в бизнес-логике).

### 2.1.2 Microsoft Component Object Model

Microsoft COM (Component Object Model – объектная модель компонентов) – межплатформенная объектная система распределенного типа для создания взаимодействующих между собой компонентов (т.н. COM-объектов) двоичного (переведенного в двоичный машинный код) ПО. Данная технология является фундаментальной технологией ActiveX, Microsoft OLE, DirectX и др.

COM-объекты могут располагаться как в рамках одного приложения, так и разных, на локальном компьютере или в удаленной сети; возможность их использования не ограничивается способом реализации приложений.

Доступ к COM-объекту достигается с помощью использования интерфейсов и методов (функций или процедур, предоставляемых тем или иным интерфейсом), которые этот объект поддерживает. Помимо этого, одним из главных требований COM является то, что доступ к COM-объектам происходит только через обращение к указателям на их интерфейсы. Соответственно, объектную модель компонентов могут поддерживать лишь те языки программирования, которые предоставляют возможность использования указателей и вызова функций через указатели.

Кроме того, COM требует, чтобы единственным способом получить доступ к методам интерфейса являлся указатель на данный интерфейс (т.е. чтобы прямой доступ к данным объекта отсутствовал).

Количество интерфейсов конкретного COM-объекта не ограничивается. Схематическое изображение структуры COM-объекта приведено на рисунке 2.1. Каждый интерфейс поддерживает как минимум три стандартных и неограниченное множество пользовательских методов.

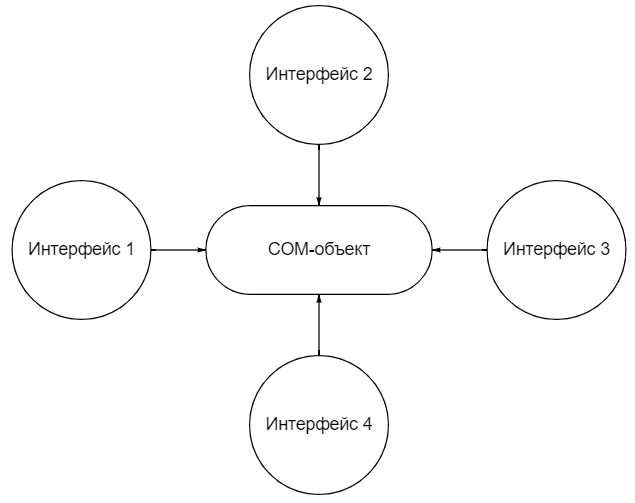


Рисунок 2.1. Структура объекта COM.

### 2.1.3 Типы взаимодействия программ

Объектная программа – это программа, классы и компоненты которой, взаимодействуя между собой, реализуют объектную модель. Благодаря использованию объектному подхода существенно снижается сложность проектирования программных систем, состоящих из десятков и сотен различных классов.

Проектирование объектной модели – это описание классов, необходимых для обеспечения требуемой функциональности системы, и установление отношений между полученными классами, поскольку эти отношения и определяют способы взаимодействия между объектами.

Перечислим основные виды межклассовых отношений в объектной модели:

1. Отношения типа «часть-целое» («has-a» в зарубежной литературе) или агрегация – в этом случае объект одного класса включает в себя объекты других классов в качестве своих составляющих. Существует строгая и нестрогая агрегация, которая определяется возможностью независимого существования главного и подчиненных объектов.

2. Обобщение («is-a») – о таких отношениях говорят, если имеется один или несколько классов, которые являются более общими по отношению к другим классам. Реализация происходит через использование механизмов наследования в объектно-ориентированных языках программирования.

3. Ассоциация – используется в тех случаях, когда невозможно конкретно установить отношения между объектами.

4. Использование и др.

## 2.2 OLE-автоматизация

Как уже ранее упоминалось, одной из технологий, поддерживающих объектную модель компонентов COM, является Microsoft OLE Automation.

Впервые в истории термин OLE (Object Linking and Embedding – связывание и внедрение объектов) появился в 1991 году и был связан с взаимодействием между программами пакета Microsoft Office (в частности, Word и Excel). Затем в 1993 году появилась вторая версия технологии OLE 2.0, и значение термина OLE разрослось, включив в себя и некоторые другие технологии с поддержкой COM. В 1996 году часть OLE, относящаяся к поддержке технологий сети Интернет, получила название ActiveX, и через некоторое время почти все технологии OLE стали относить к ActiveX (исключая лишь технологию составных документов, которая относилась к программам Microsoft Office). Однако некоторые OLE-технологии вообще не имеют отношения ко внедрению и связыванию объектов – в частности, одной из таких технологий является OLE Automation (OLE-автоматизация, к настоящему времени переименованная в Автоматизацию). Главная функция OLE Automation – обеспечить совместную работу компонентов и приложений, реализованных на различных языках программирования – никак не связана с составными документами или офисными программами.

Автоматизация – механизм взаимодействия между программами, основанный на подмножестве модели COM (изначально реализованном для скриптовых языков), который дает возможность одному приложению управлять объектами другого, либо предоставлять управление к своим объектам. Все программы, использующие автоматизацию (т.н. контроллеры автоматизации), должны содержать производные от IDispatch COM-интерфейсы, которые работают с определенными типами данных, получившими название типов автоматизации. В этой модели приложение, предоставляющее объекты автоматизации, считается сервером, а контроллер автоматизации, получающий доступ к этим объектам, – клиентом. Согласно своему названию, объекты автоматизации, хотя и могут использовать OLE, но не всегда это делают.

Некоторые языки программирования (такие как C++, Visual Basic или Delphi), упрощают использование объектов автоматизации, скрывая сложность их реализации за более простыми и понятными синтаксическими языковыми конструкциями. В целом, автоматизацию возможно реализовать с использованием большинства популярных языков программирования (C, C++, C#, Python, Java (используя jSegue или j-Interop), VBA, PHP, Perl, Delphi и мн.др.).

Для того, чтобы была возможность работать с объектами автоматизации из других приложений, разработчик целевой программы должен подробно задокументировать ее объектную модель. Объектная модель Microsoft Office публично задокументирована (в виде библиотеки типов), кроме того, есть множество других приложений, разработчики которых также позаботились о решении данного вопроса.

Поддержка компонентов и средств OLE, как правило, обеспечивается самой ОС.

## 2.3 Прямой доступ к данным OLE-сервера из стороннего приложения

Система планирования и управления производством Zenith SPPS может выступать в роли OLE-сервера и рассчитана на то, чтобы сторонние приложения обращались к ней, осуществляя доступ к данным напрямую, используя функции OLE Automation только в случаях необходимости (если напрямую какие-то данные получить программно достаточно сложно). Прямой доступ к данным системы осуществляется путем параллельного соединения с внутренней базой данных, используя OLE-свойство DBParameters, содержащее в себе строку для соединения с БД (Connection String) со всеми необходимыми для подключения параметрами.

Программно непосредственный доступ к данным реализуется с использованием навигационных методов и свойств или реляционным путем, т.е. описывая желаемый результат в виде запросов на языке SQL.

Язык программирования Delphi и среда разработки, поддерживающая его, позволяет осуществить подключение к внутренней базе данных Zenith SPPS Microsoft Access через компоненты ADO (ActiveX Data Objects – объекты данных ActiveX) и также дает возможность использовать реализованный функционал OLE-автоматизации, поскольку поддерживает совместимый с ней тип OLEVariant, через который можно обращаться к методам OLE-сервера.

### 2.3.1 Навигационные методы

Использование навигационного способа к данным подразумевает обращение к отдельным записям и выполнение различных операций с ними. Чаще всего этот способ работы с данными используют применительно к локальным базам данных, поскольку приложение при этом получает весь набор записей, что может серьезно перегрузить сеть.

Обращение к данным происходит с помощью указателя, который ссылается на выбранную в данный момент запись. Такой указатель содержится в любом наборе данных. В языке программирования Delphi существуют специальные компоненты для работы с данными, позволяющие перемещаться по данным путем изменения текущего указателя, такие как Table (таблица) и Query (запрос). Отличие Table от Query заключается в том, что Table может работать только с одной таблицей, а в Query можно использовать соединения и объединения из нескольких таблиц.

Работа с навигационными методами может осуществляться как программно, так и с помощью компонентов.

Реализован ряд методов и свойств объектов Table и Query для навигации по таблице. Приведем некоторые наиболее часто используемые:

1. Методы First, Last, Next, Prior – позволяют получить первую/последнюю/следующую или предыдущую соответственно запись в наборе данных.

2. Свойства BOF (Begin Of File) и EOF (End Of File), указывающие на начало и конец набора записей.

3. Процедура MoveBy, с помощью которой можно перемещаться на необходимое число записей вперед или назад по таблице.

4. Методы Locate и Lookup, первый из которых позволяет найти первую запись в наборе, удовлетворяющую условию и критериям поиска, указанным в параметрах метода, и вернуть ее, а второй – проверить точное совпадение записи с условием, не возвращая ее.

5. Свойства Filter и Filtered, используя которые можно отфильтровать записи в наборе данных по достаточно несложным условиям фильтрации.

Также существует несколько свойств и методов для получения самих записей программным путем:

1. Свойство Fields – для обращения к полю в записи (как в массиве значений) по индексу (индексация начинается с 0). Например, Table1.Fields[3] обращается к четвертому по счету в наборе данных полю таблицы Table1. Может также использоваться для обращения к полю по имени (через свойство FieldName).

2. Метод FieldByName позволяет получить значение определенного поля по его имени – это достаточно удобно в случаях, если структура запроса может измениться. Например, с помощью конструкции *Table2.FieldByName(‘Surname’)* можно получить значение поля Surname из таблицы Table2.

3. Свойство FieldValues – используется по умолчанию, содержит массив полей текущей записи, обращаться к элементам которой можно через имена этих полей. Возвращает тип Variant, что является не всегда удобным.

Обращение к одному и тому же полю разными способами дает идентичные результаты и такая возможность создана лишь для удобства самих разработчиков.

### 2.3.2 Реляционный доступ к данным

Отличие реляционных методов от навигационных состоит в том, что в навигационных идет перемещение по набору данных и работа с одной текущей выбранной на данный момент записью, а в реляционных – с группой записей (т.е. даже в случае необходимости обработки одной записи, будет обработана группа из одной записи). Для получения данных в реляционном способе доступа используются SQL-запросы. Как правило, этот способ применяется при работе с распределенными и удаленными базами данных, но может использоваться и в локальных (хотя в них предпочтительнее использовать навигационные методы).

Программно реляционный доступ к данным реализуется через компонент Query.

Компонент Delphi Query является достаточно универсальным, т.к. с его помощью можно обращаться к базе данных с помощью запросов на языке SQL. Полученные данные будут представлены в виде временной таблицы. Если необходимо получить данные без внесения изменений, то в запросе используется оператор SELECT; при необходимости изменений используются операторы INSERT, UPDATE и DELETE. Запросы бывают статическими (неизменяемые, формируются и даже выполняются во время создания программы) и динамическими (выполняются в момент функционирования программы, с помощью метода Add или загружаясь из внешнего файла).

Также компонент Query позволяет строить параметрические запросы (с помощью массива параметров Params, свойства ParamCheck и метода Prepare).

Запросы в Query описываются в свойстве SQL.Text, например, с помощью следующей строки можно получить все данные из таблицы User:

Query1.SQL.Text := ‘SELECT \* FROM User’

Изменение данных производится с помощью вызова процедуры ExecSQL; получение данных – с использованием метода Open или свойства Active.

Закрывать набор данных следует методом Close, в случае применения которого никаких исключительных ситуаций подниматься не будет.

### 2.3.3 Технология ADO

Доступ к данным из стороннего приложения может реализовываться несколькими способами: через API СУБД, через клиентский API (или клиентские объекты COM), через т.н. универсальные механизмы доступа (драйверы или провайдеры). У каждого из перечисленных способов есть свои достоинства и недостатки, выбор того или иного ситуативен и зависит от поставленной задачи. В частности, универсальные механизмы доступа удобно использовать в случаях, когда может потребоваться смена СУБД, к которой обращается программа, но в них невозможно учесть специфику каждой конкретной СУБД и их использование понижает производительность программы.

ADO, наряду с ODBC, BDE (Borland Database Engine), OLE DB (Object Linking and Embedding Database) и др., относится к универсальным механизмам доступа к данным. ADO и OLE DB являются разработкой Microsoft, частью Microsoft UDA (Microsoft Universal Data Access – универсального механизма доступа к данным). Универсальность заключается в том, что OLE DB и ADO совместно организуют доступ к самым различным источникам данных как реляционным, так и не поддерживающим реляционную модель.

Механизм ADO построен на основе использования COM-интерфейсов, предоставляемых OLE DB. OLE DB осуществляет доступ к данным посредством применения стандартных интерфейсов, которые не зависят ни от места нахождения данных, ни от их типа.

ADO (ActiveX Data Object) представляет собой совокупность программных модулей, реализующих API для доступа к данным, которую можно использовать в клиентских приложениях. Технология ADO, как понятно из ее названия, основана на программной платформе ActiveX. В качестве низкоуровневых интерфейсов доступа к данным в ADO используются компоненты OLE DB. Обозначенный механизм получения доступа к данным изображен на рисунке 2.2.

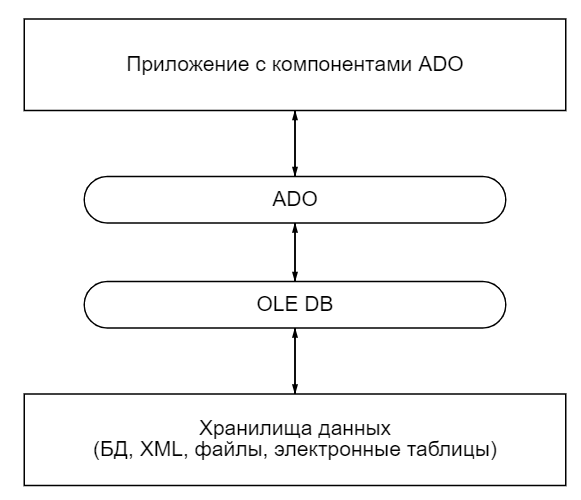


Рисунок 2.2. Получение доступа к данным из приложения с помощью ADO.

Данные, получаемые посредством ADO, представляются в объектно-ориентированном виде.

Компонентами объектной модели ADO являются:

* Connection – подключение к источнику данных;
* Recordset – набор данных от источника данных и Record – одна его запись;
* Command – для выполнения команд и параметрических SQL-запросов;
* Fields – столбцы таблицы БД;
* Parameters – набор параметров параметрического запроса;
* Properties – свойства объекта;
* Errors – ошибки в программе, и др.

Для каждого типа хранилищ данных существует свой провайдер ADO, который знает расположение данных, каким образом к ним обращаться, как их интерпретировать и возвращать сторонним приложениям. Провайдер выбирается из списка доступных в ОС в процессе установки соединения через ADO-компоненты.

Технология ADO также имеет поддержку транзакций.

ADO-компоненты поддерживаются некоторыми высокоуровневыми языками программирования (JScript, Visual Basic, Delphi и др.). Компания Microsoft включила их в состав платформы .Net, изменив название на ADO.NET.

#### 2.3.3.1 Компоненты Delphi для поддержки ADO

Среда разработки Delphi имеет множество высокоуровневых компонентов, обеспечивающих и упрощающих работу с базами данных при разработке программ.

Состав компонентного набора определяется версией Delphi, а также видом самого приложения. В последних его версиях существует возможность создания программ для .Net и Win32.

В современных средах разработки 32-разрядные приложения Delphi для Windows называются VCL Form Application (Visual Component Library – библиотека визуальных компонентов). А компоненты работы с БД расположены в меню компонентов на вкладках Data Access (организация доступа к данным), Data Controls (отображение данных), dbExpress (реализация технологии dbExpress), dbGo (компоненты ADO) и т.д.

Компоненты для доступа к данным и работы с данными выполняют функции соединения с БД, выполнения запросов, получения и отображения результатов, изменения данных и мн.др. Выводимые данные могут относиться к различным форматам, в том числе и произвольным.

Все компоненты Delphi для работы с БД относятся к одной из трех больших групп:

1. Источники данных (или, иначе, Data Sources) – невизуальные компоненты, соединяющие визуальные компоненты данных и наборы данных. Примером является TDataSource, расположенный на вкладке Data Access.

2. Визуальные компоненты (Data Controls) – предназначены для пользовательского просмотра и редактирования данных. К ним относятся DBEdit, DBImage, DBListBox, DBCtrlGrid и др.

3. Наборы данных (Data Sets) – невизуальные компоненты для обеспечения доступа к данным; наиболее часто используются Table и Query.

Компоненты ADO также относятся к невизуальным компонентам получения доступа к данным. Существует несколько компонентов Delphi для применения технологии ADO: ADOConnection (обеспечивает соединение с базой данных через свойство ConnectionString, также дает возможность осуществления транзакций), ADOQuery (формирует запросы к БД, использующие операторы SELECT, INSERT, UPDATE или DELETE), ADOTable (компонент для доступа к таблице), ADODataSet (позволяет визуальным компонентам работать с полученным набором данных), ADOCommand (для выполнения не возвращающих результат SQL-инструкций) и ADOStoredProc (для вызова хранимой процедуры, расположенной на сервере БД, позволяет возвращать результирующие данные). Как было сказано ранее, все компоненты для поддержки ADO располагаются в последних версиях Delphi на вкладке dbGo.

Компоненты ADO очень удобно применять для осуществления работы с базами данных Microsoft Access, поскольку драйвер, используемый для подключения к такой базе, создан корпорацией Microsoft, как и сама технология ADO, и потому является для них более надежным, чем драйвера сторонних производителей.

## 2.4 Выбор программно-технической среды реализации

В качестве языка программирования для разработки был выбран Delphi (являющийся диалектом языка Object Pascal), поскольку он является объектно-ориентированным и поддерживает весь необходимый для решения поставленной задачи набор технологий (ADO, OLE DB, OLE-автоматизацию, прямой доступ к данным и др.).

В качестве интегрированной среды разработки (IDE) была выбрана Embarcadero Delphi 10.4.2 Sydney в версии Community Edition, часть программного пакета Embarcadero RAD Studio, поскольку она является бесплатным решением с ограниченной лицензией для некоммерческого использования и предназначена для быстрой разработки прикладных программ RAD (Rapid Application Development - концепции быстрой разработки приложений) под различные ОС (Microsoft Windows, Linux, Android, MacOS X и iOS), что довольно удобно. Данная версия обладает некоторыми преимуществами и новыми возможностями по сравнению с более старыми, но в рамках решения данной задачи они существенной роли не играют (кроме, возможно, улучшений быстродействия приложений на VCL).

### 2.4.1 Требования к аппаратному обеспечению и программно-технической среде

Общий список требований, минимально необходимых для работы MES-системы Zenith SPPS и IDE Embarcadero Delphi 10.4 Community Edition, выглядит следующим образом:

1. Двухъядерный или более мощный процессор с частотой 1,8 ГГц или более.

2. Минимум 2 Гб оперативной памяти (ОЗУ).

3. Не менее 6 Гб свободного дискового пространства.

4. Видеокарта с поддержкой DirectX 11Поддерживающая Direct X и разрешением не менее 1024x768 и соответствующий монитор.

5. Принтер для вывода информации в графическом режиме.

6. В качестве используемой ОС необходима Windows 10 (желательно на русском языке для упрощения настройки и установки Zenith SPPS).

7. Наличие в системе установленного драйвера ADO (включен по умолчанию во все современные версии семейства ОС Microsoft Windows).

8. Длина переменной среды PATH 1830 символов или менее (необходимо для настройки RAD Studio).

9. Предоставление прав администратора системы при установке обеих программ.

## 2.5 Вывод по главе

Организация взаимодействия между программами и доступ к базам данных из сторонних приложений происходит с помощью промежуточного ПО и в соответствии с различными стандартами. Одним из самых популярных стандартов для организации межпрограммного (или межпроцессного) взаимодействия является объектная модель компонентов COM. В соответствии с ней, доступ к любым объектам осуществляется посредством обращения к COM-интерфейсам через их публичные методы.

Microsoft OLE Automation - одна из технологий, поддерживающих стандарт COM. С помощью нее приложение может управлять объектами другого приложения, либо предоставлять управление к своим объектам. Поддержка OLE-автоматизации обеспечивается на уровне операционной системы.

В настоящей выпускной квалификационной работе система Zenith SPPS выступает в роли OLE-сервера, к которому необходимо подключиться из программной надстройки путем использования функций OLE. Также необходимо организовать непосредственный доступ к данным, поскольку основная работа с системной базой данных будет происходить именно таким способом.

Прямой доступ к данным из приложения может осуществляться навигационными способами (переключением между записями в наборе данных) либо реляционными методами (с помощью запросов и работы с множествами данных). Оба способа реализуются как программно, так и на уровне компонентов.

Для доступа к данным (в данном случае, базе данных MS Access) из Delphi существуют ADO-компоненты. ADO – технология доступа к данным от Microsoft (в некотором роде надстройка над OLE DB), основанная на использования COM-интерфейсов, предоставляемых OLE DB (низкоуровневая технология доступа к данным также от Microsoft).

# Глава 3 Реализация и тестирование программной надстройки для объединения технологических операций

## 3.1 Подготовка и настройка программной среды реализации

До начала работы необходимо установить пробную версию программы Zenith SPPS с ее официального сайта по ссылке <http://www.zspps.ru/index-php/mes-download1.html>. Перед загрузкой установочного файла потребуется заполнить форму на сайте данными организационного характера, а затем нажать на кнопку «Скачать пробную версию».

Установка пробной демонстрационной версии должна осуществляться с предоставлением прав администратора, в противном случае программа может работать некорректно. Также следует разрешить исполняемому файлу Zenith SPPS запуск от имени администратора в свойствах файла на вкладке «Совместимость». Вид окна со свойствами продемонстрирован на рисунке 3.1.

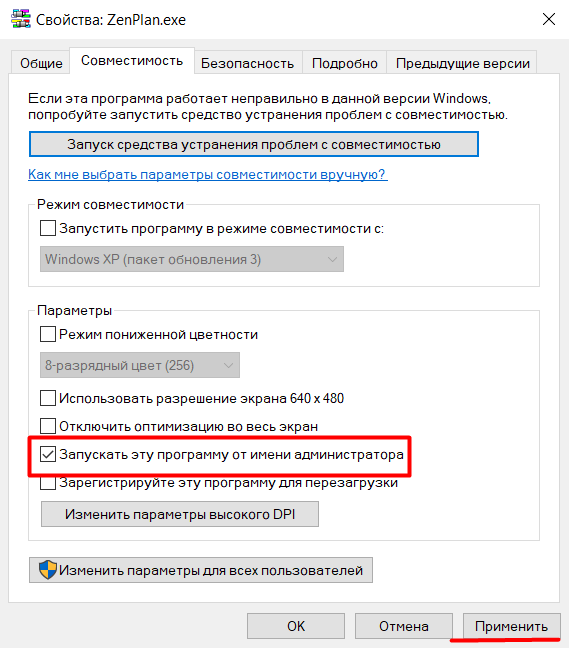


Рисунок 3.1. Настройка запуска системы Zenith SPPS от имени администратора.

Помимо этого, IDE, с помощью которой осуществляется разработка (в данном случае, Embarcadero Delphi 10.4 Community Edition), также должна быть запущена от имени администратора, поскольку в противном случае при попытке подключиться к программе будет выдаваться ошибка (скриншот окна с ошибкой изображен на рисунке 3.2). Запуск с правами администратора можно настроить таким же образом, как и для исполняемого файла системы Zenith SPPS.

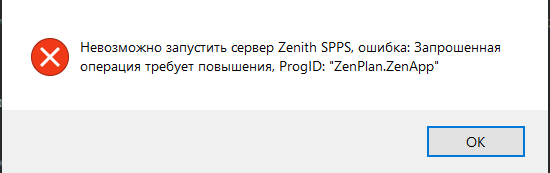


Рисунок 3.2. Вид окна с ошибкой о недостаточности прав для подключения к системе Zenith SPPS из среды разработки.

Перед началом работы необходимо, согласно «Руководству разработчика интерфейса» [1], зарегистрировать основное приложение системы ZENPLAN.EXE в качестве OLE-сервера. Для этого необходимо запустить окно команд ОС Windows «Выполнить» (нажав сочетание клавиш Win+R, через меню Пуск или Поиск программ), и в открывшемся окне ввести строку, содержащую путь к исполняемому файлу приложения и команду */regserver*:

"C:\Program Files (x86)\Zenith SPPS\Bin\ZenPlan.exe" /regserver

Если путь содержит пробелы, он должен быть заключен в двойные кавычки.

Отмена регистрации сервера происходит таким же образом, только вместо команды */regserver* нужно использовать команду */unregserver*.

Также для того чтобы удобно отображались информационные сообщения в системе, следует перейти в основном модуле системы Zenith SPPS в пункт Главного меню Сервис -> Параметры программы, и на вкладке «Общее» открывшегося окна выбрать из списка с помощью стрелочек пункт «Сообщения пользователя из обл.уведомлений», как показано на рисунке 3.3.

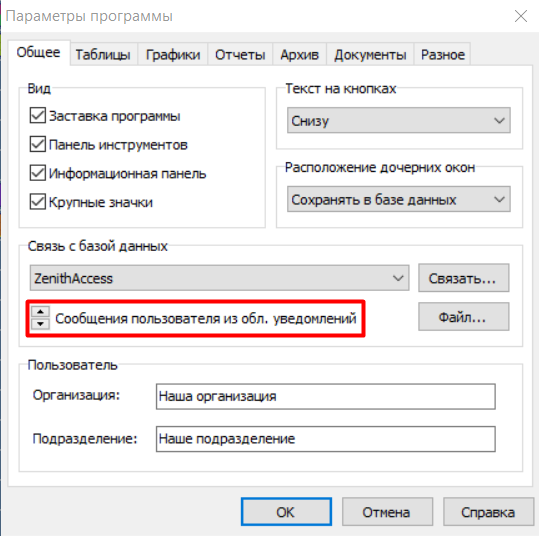


Рисунок 3.3. Настройка вывода пользовательских сообщений в системе Zenith SPPS.

## 3.2 Создание объекта ZenApp и подключение к внутренней БД

Далее для создания надстройки требовалось подключение к основному модулю MES-системы Zenith SPPS и ее внутренней базе данных. Поскольку при решении задачи использовалась пробная версия программы, ее внутренняя БД – это база данных Microsoft Access.

Чтобы подключиться программно с помощью Delphi к Zenith SPPS и его внутренней БД, необходимо создать объект ZenApp типа OLEVariant с помощью функции *CreateOleObject* из библиотеки System.Win.ComObj, которая создает экземпляр объекта OLE Automation. После этого с помощью OLE-свойства AppHandle получить дескриптор окна системы Zenith SPPS, и назначить его дескриптором окна приложения надстройки, чтобы программная надстройка работала с MES-системой как одно приложение.

Затем с помощью свойства DBParameters получить строку соединения (Connection String), содержащую все необходимые параметры для подключения к БД, и задать значение свойства Connected равным истине для установления соединения с набором данных.

Фрагмент кода, содержащий подключение к системе Zenith SPPS и его базе данных MS Access показан на рисунках 3.4 и 3.5.

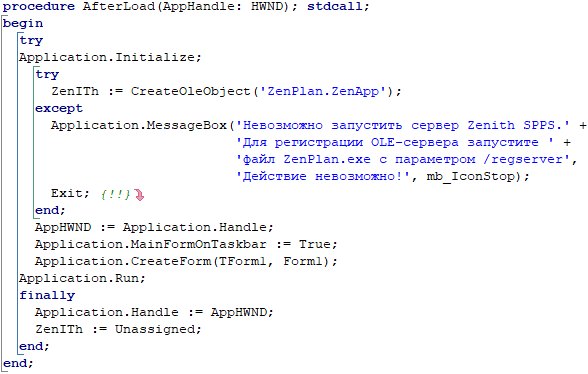


Рисунок 3.4. Программное подключение к системе Zenith SPPS.

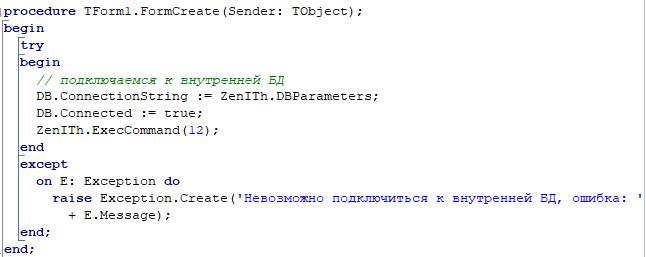


Рисунок 3.5. Программное подключение ко внутренней БД системы Zenith SPPS.

## 3.3 Реализация объединения технологических операций

### 3.3.1 Получение атрибутов выбранной в Zenith SPPS операции

Первым делом необходимо получить идентификатор выбранной пользователем в Зените операции, чтобы затем с его помощью определить остальные атрибуты – рабочее место, вид работы и обрабатываемую деталь. В целевой таблице PlanTask первичный ключ является составным из двух полей (TaskID, PartID), то есть сам TaskID в общем случае не является уникальным, но дублироваться он может только в двух случаях:

1. Если технологическая операция представляет собой сборочную операцию.

2. Либо если технологическая операция является объединением технологических операций.

Но и в том, и в другом случае из TaskID можно однозначно определить рабочее место (поскольку и объединения, и сборки выполняются в рамках одного рабочего места и в системе Zenith SPPS представляют собой не множество операций на графике разных цветов, а одну многоцветную, просто более длинную), вид работы (поскольку сборки, внутри которых содержатся операции, относящиеся к разным видам работ, в решении данной задачи не рассматриваются), а также деталь или множество деталей, над которыми осуществляется операция. Изображение сборочной операции в MES-системе Zenith SPPS показано на рисунке 3.6.



Рисунок 3.6. Сборочная операция на Графике загрузки рабочих мест в MES-системе Zenith SPPS.

Определим TaskID при помощи OLE-свойства CurrentTaskID, остальные поля – запросом непосредственно к БД.

Необходимо также проверить в случае, если выбрана сборочная операция (поле Assembly = true и во всех операциях с тем же TaskID, кроме одной, в поле TaskName будет стоять звездочка \*), что все входящие в нее операции относятся к одному виду работ. Сборочные операции, относящиеся к различным видам работ, в данной выпускной квалификационной работе не рассматриваются.

Поле Assembly истинно также для объединений, только в них TaskName будет заполнен для всех операций так, как если бы они были обособлены друг от друга.

### 3.3.2 Вывод возможных для объединения операций

Далее необходимо вывести пользователю список возможных для объединения операций (кандидатов объединения).

Поскольку поиск таких операций необходимо осуществлять одновременно в двух направлениях (более ранние относительно выбранной, более поздние относительно выбранной), при проверках могут возникнуть ситуации, когда по отдельности какие-то операции с выбранной объединить можно, а вместе они нарушат технологическое расписание. В этом случае более правильным кажется ориентироваться на выбор пользователя, т.е. детальные проверки на возможность объединения определенных операций необходимо производить только после того, как пользователь осуществит свой выбор.

Однако для того чтобы не выводить весь список операций в расписании по нужному рабочему месту, минимальный набор адекватных проверок все же должен происходить до вывода списка пользователю. Заранее необходимо проверить, чтобы разрыв между соседними операциями был строго меньше, чем константа MinOperationLength из таблицы PlanConsts (минимальная продолжительность операции в минутах). Это нужно, поскольку если разрыв между операциями больше или равен минимальной продолжительности, то при дальнейшей перепланировке система может запланировать на это время какую-то другую операцию (либо же этот простой может быть организован с определенной целью, неизвестной разработчику).

Также до вывода пользователю списка можно ограничить количество операций, выбрав только те из них, которые начинаются не раньше, чем заканчивается предыдущая по техпроцессу для выбранной операции и заканчивались не позже, чем начинается последующая.

Определение текущей выбранной в Zenith SPPS операции и вывод операций, являющихся кандидатами в объединение относительно нее, осуществляется при нажатии на кнопку «Показать список доступных (с учетом выделенной) операций «Показать список доступных (относительно выделенной на Графике) операций» и реализован в процедуре FormShowButtonClick. Код процедуры FormShowButtonClick приведен в составе полной реализации кода программной надстройки в [Приложении 1](#_Приложение_1_Листинг).

### 3.3.3 Осуществление проверок на возможность объединения

*Главные критерии*, по которым будут осуществляться проверки корректности объединения технологических операций в системе Zenith SPPS:

1. Объединение операций не должно нарушить производственное расписание. Следовательно, все операции, входящие в состав объединенной, должны начинаться не раньше, чем закончится самая последняя из предыдущих операций, зависящих от них, и заканчиваться не позже, чем начнется самая ранняя из последующих.

2. Однотипность операций – т.е. все объединяемые операции должны относиться к одному и тому же виду работы.

Помимо этого, они все должны располагаться рядом в рамках одного рабочего места и разрыв между ними должен быть меньше минимальной продолжительности операции в системе, а также у них не должны совпадать номера деталей, поскольку в этом случае объединение произойти не сможет в связи с ограничением первичного ключа целевой таблицы PlanTask.

В связи с возможностью образования сборочных операций и объединений в расписании поиск по таблице PlanTask в рамках необходимого рабочего места происходит определенным образом:

* «вертикально» вдоль всего списка операций, отсортированных по времени начала операции, если операции одиночные (т.е. поле Assembly = false);
* или «горизонтально» в пределах одного TaskId для всех относящихся к нему деталей, если это сборка или объединение (т.е. поле Assembly = true).

Когда пользователь выбирает в списке какие-то операции, другие из них, которые по итогам проверок невозможно объединить с отмеченным пользователем множеством, блокируются и повторная возможность их выбора появляется только в том случае, если сняты галочки с тех операций, которые препятствовали их объединению.

Также предусмотрено, чтобы в случае выбора операции, относящейся к объединению/сборке, выбирался либо весь список входящих в ее состав операций, либо ни одна из них.

С учетом всего вышесказанного, *алгоритм проверки корректности объединения* выбранных операций следующий:

1. Пользователь выбирает какую-то операцию в списке.

2. Операции проверяются, начиная с крайней в списке выбранных операции противоположного относительно выбора направления (если выбрана более ранняя относительно текущей выбранной в системе операции, тогда проверяем начиная с наиболее поздней, и наоборот).

3. Осуществляется проверка уникальности деталей (PartID). В случае, если деталь уже встречалась среди предыдущих выбранных операций, операция блокируется и все последующие операции вместе с ней. В случае, если эта деталь относится к сборке/объединению, блокируется вся сборка/объединение.

4. Производится проверка всех предыдущих (поле RecordPosition меньше на 1, PartID тот же) и следующих (поле RecordPosition больше на 1, PartID тот же) операций для кандидата объединения, чтобы технологическое расписание не было нарушено (все предыдущие заканчиваются не позже, чем начинается самая ранняя из объединяемых операций, а все следующие начинаются не раньше, чем заканчивается самая поздняя из них). В случае, если проверка не пройдена, операция блокируется и также блокируются все последующие. Если не прошедшая проверку операция является частью сборки/объединения, блокируются все операции данной сборки/объединения.

5. Если проверка пройдена, операция отмечается галочкой на списке и таким же образом проверяется следующая в соответствующем направлении, пока программа не дойдет до операции, отмеченной пользователем, или не вылетит на одной из проверок.

Проверки происходят в методе FormListBoxClickCheck, при выборе пользователем операций в списке возможных кандидатов для объединения. Код процедуры FormListBoxClickCheck приведен в составе полной реализации кода программной надстройки в [Приложении 1](#_Приложение_1_Листинг).

### 3.3.4 Объединение операций

В случае успешного прохождения всех проверок, пользователю в интерфейсе надстройки станет доступна кнопка «Объединить выбранные операции».

При объединении нескольких однотипных операций в одну их TaskID становится равным минимальному TaskID из них всех, поле Assembly обретает значение истины, поля Start и Finish (а также StartDateTime и FinishDateTime соответственно) для всех становятся равными началу самой ранней в объединении операции и завершению самой поздней. Поле Processing (фактическая продолжительность операции) является вычисляемым полем и в общем случае равно сумме продолжительностей всех объединяемых операций. Поле Preparation (подготовительно-заключительное время) не суммируется по всем объединяемым операциям, а выбирается максимальное, поскольку подготовительно-заключительное время не зависит от объема операции.

Изменения в записи происходят в процедуре FormUnionButtonClick. Код процедуры FormUnionButtonClick приведен в составе полной реализации кода программной надстройки в [Приложении 1](#_Приложение_1_Листинг).

После того, как запрос ко внутренней БД на изменения выполнится, необходимо подождать некоторое время (примерно 4 секунды для MS Access), и только после этого обновить интерфейс системы. В противном случае внесенные данные рискуют не отобразиться в программе (с связи с особенностями драйвера СУБД MS Access).

### 3.3.5 Вид интерфейса для объединения операций

При разработке интерфейса программы, осуществляющей объединение операций, наиболее важным критерием было отобразить для пользователя внутренний механизм работы проверок для выбираемых операций таким образом, чтобы он понимал связи и зависимости операций друг относительно друга и у него не возникало вопросов, почему какую-то операцию можно выбрать, но после этого выбора выбор других может быть невозможен. Это было достаточно сложно, поскольку сама система Zenith SPPS не поддерживает множественный выбор на Графике загрузки рабочих мест – соответственно, невозможно одновременно показать пользователю на Графике наглядно местонахождение и возможные пересечения всех предыдущих и следующих операций относительно выбранных кандидатов в объединение.

Итоговая версия интерфейса программы, реализующей объединение операций, показана на рисунке 3.7.

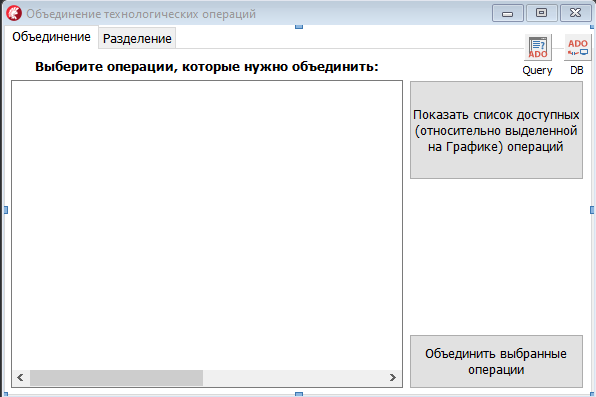


Рисунок 3.7. Интерфейс объединения технологических операций.

## 3.4 Программная реализация задачи разделения технологических операций

Задача объединения технологических подразумевает также и разделение операций при необходимости.

### 3.4.1 Вывод разъединяемых операций

Для вывода всех операций, входящих в состав объединения, определяется текущая выбранная в системе Zenith SPPS операция также с помощью OLE-свойства CurrentTaskID. Если выбранная операция не относится к объединению, программа выдаст предупреждение. Разделять сборочные операции не представляется возможным, поскольку не понятно в таком случае, каким образом восстанавливать для них значение поля TaskName, которое в сборочных операциях для всех, кроме главной, равно ‘\*’. Также такое разделение не является корректным по смыслу.

Вывод списка разделяемых операций происходит при нажатии на кнопку «Проверить корректность разделения выбранной на Графике операции» в процедуре DeformShowButtonClick. Код процедуры DeformShowButtonClick приведен в составе полной реализации кода программной надстройки в [Приложении 1](#_Приложение_1_Листинг).

### 3.4.2 Разъединение операций

Важной особенностью процесса разъединения технологических операций является то, что поскольку время выполнения операций затирается при их объединении (суммируется определенным образом, описанным в [п.3.3.4](#_3.3.4_Объединение_операций)), то его необходимо восстанавливать.

Восстановление фактического времени выполнения операции происходит по-разному в зависимости от объема вида работ, на который назначается время выполнения. Время выполнения операции может быть назначено на:

* единицу объема или штучное – в этом случае общее время выполнения вычисляется как произведение объема работ и времени выполнения данного вида работ, к которому прибавляется подготовительно-заключительное время;
* на всю операцию – тогда просто суммируется время выполнения операции и подготовительно-заключительное время;
* на указанное значение – в этом случае объем работ умножается на время выполнения и это значение делится на указанное значение единиц; к нему также прибавляется подготовительно-заключительное время.

Объем вида работ, на который назначается время выполнения, находится в поле Portion таблицы PlanGrpl. Определить его можно по номеру детали и виду работы (поля PartID и GroupID соответственно). В этой же таблице находятся поля InitProcessing и InitPreparation, значения которых необходимо использовать в случае, если нормативное время выполнения и нормативное подготовительно-заключительное время операции соответственно отсутствуют в таблице PlanTask. Объем операции необходимо брать либо из таблицы PlanTask (поле TaskVolume), либо, в случае его незаполненности, из таблицы PlanPart (поле PartVolume), определив его по детали (полю PartID). Данные поля могут содержать пустые значения, если внутренняя база данных старой версии (как в случае с пробной версией программы). Запрос, вычисляющий время выполнения операции и подготовительно-заключительное время приведен на рисунке 3.8.

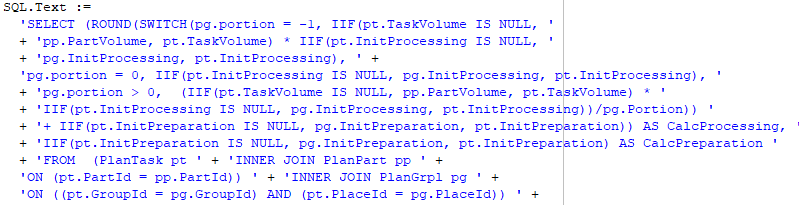


Рисунок 3.8. SQL-запрос на вычисление значений полей Processing и Preparation для разделяемых операций.

Также, поскольку старые значения TaskID тоже потеряны, необходимо генерировать новые TaskID для отделяемых операций. Делается это с помощью OLE-свойства NewIDValue. Поскольку назначение TaskID операциям будет происходить в цикле, а свойство NewIDValue производит достаточно объемные вычисления, будет удобнее использовать его 1 раз, а далее просто прибавлять каждый раз 1.

Помимо Processing, TaskID и Preparation, необходимо также задавать значения полям Start (количество минут рабочего времени в оперативном плане до начала данного задания), Finish (Количество минут рабочего времени в оперативном плане до окончания данного задания), StartDateTime и FinishDateTime (время начала и окончания задания соответственно), Assembly. Поля Start и Finish вычисляются с помощью времени начала объединенной операции и вычисленного значения поля Processing (алгоритм вычисления которого был рассмотрен ранее в этом подразделе), а поля StartDateTime и FinishDateTime вычисляются с помощью OLE-свойства PosToTime, которое возвращает астрономическое время (тип данных DateTime), соответствующее количеству минут рабочего времени, прошедших от начального интервала планирования (значениям полей Start и Finish).

Однако у разделения операций в рамках данной задачи есть некоторая особенность: поскольку TaskID и время выполнения операций затирается при объединении и восстановить его нет никакой возможности исходя из имеющейся структуры внутренней БД, при разделении операции находятся не в том порядке друг относительно друга, в котором находились изначально (восстановить исходный порядок следования операций не представляется возможным, поэтому в данном случае операции будут располагаться по возрастанию номера операции).

Разъединение операций и сопутствующие ему вычисления осуществляются в процедуре DeformDivideButtonClick. Код процедуры DeformDivideButtonClick приведен в составе полной реализации кода программной надстройки в [Приложении 1](#_Приложение_1_Листинг). Перед обновлением интерфейса системы необходимо также подождать 4-5 секунд.

### 3.4.3 Вид интерфейса разъединения операций

На рисунке 3.9 изображен интерфейс программы, реализующей разделение операций.

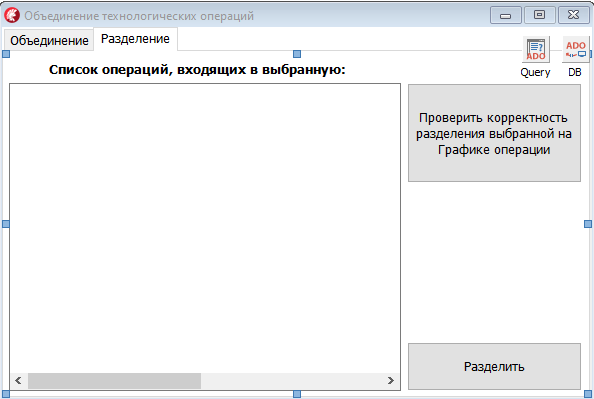


Рисунок 3.9. Интерфейс разъединения технологических операций.

## 3.5 Функциональное тестирование программной надстройки

Для того, чтобы запустить программную надстройку для объединения технологических операций, необходимо выбрать пункт «Сервис» главного меню системы Zenith SPPS, затем вкладку «Инструментальные средства». В открывшемся окне нажать на кнопку «Добавить», и для добавленного элемента выбрать путь расположения и название, под которым надстройка будет отображаться в системе. Затем нажать «ОК». Вид настроечного окна изображен на рисунке 3.10.

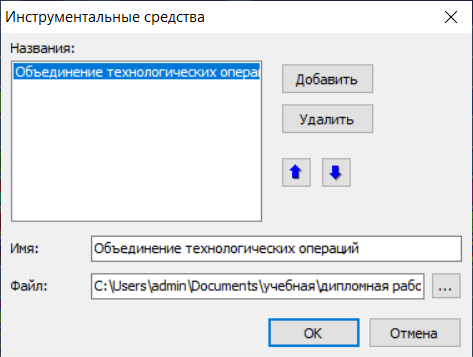


Рисунок 3.10. Добавление программной надстройки для объединения технологических операций в MES-систему Zenith SPPS.

После этого ее можно будет увидеть отдельной вкладкой и запустить через пункт главного меню системы «Сервис», которая будет иметь то название, которое было задано при добавлении.

Перейдем к тестированию части программы, осуществляющей объединение технологических операций.

Всего получилось насчитать 4 различных случая, которые могут встречаться в процессе объединения операций:

1. Объединяются одиночные операции.

2. Одиночная операция объединяется с объединением/сборкой.

3. Объединение/сборка объединяется с одиночной операцией.

4. Объединения/сборки объединяются между собой.

Возьмем для примера ряд операций на рабочем месте «Место упаковки №2». Рассмотрим подробнее каждый из четырех случаев.

*1. Объединение одиночных операций.*

На рисунке 3.11 показано расположение операций на выбранном рабочем месте. Операции одиночные, расположены друг за другом.

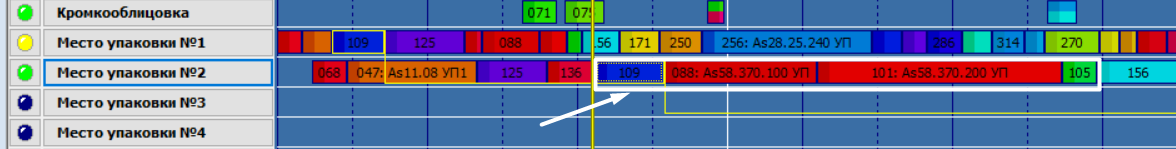


Рисунок 3.11. Одиночные операции на Графике загрузки рабочих мест.

Попробуем объединить операции со «109: As58.055 УП1» по «105: As56.10 УП2cтанд.» (для удобства здесь и далее будем называть их 109 и 105 операциями соответственно). Для этого запустим программную надстройку для объединения технологических операций, выбрав во вкладке «Сервис» Главного меню системы пункт «Объединение технологических операций».

Выбираем в системе на Графике загрузки рабочих мест любую из операций, которые хотим объединить (выберем 109) кликом мыши по операции. Далее в окне программной надстройки нажимаем на кнопку «Показать список доступных (относительно выделенной на Графике) операций». В области слева от кнопки появится список операций, которые можно объединить с выбранной на Графике. В открывшемся списке выбираем операции 109, 088, 101 и 105 (можно выбрать 105, тогда сразу выберутся и все остальные операции между 109 и 105). Вид списка с выбранными операциями показан на рисунке 3.12.

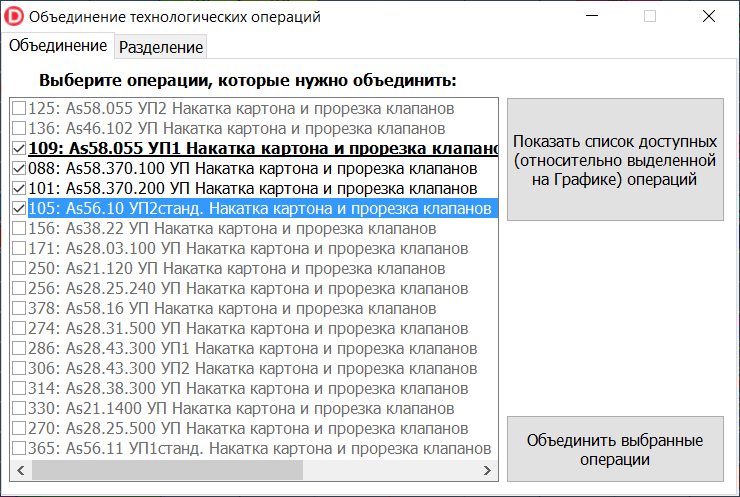


Рисунок 3.12. Выбор операций для объединения в Случае 1.

Проверим корректность выбора операций. Если попытаться выбрать 156 операцию, она заблокируется, потому что ее предыдущая операция на Графике пересекает 109 (как видно на рисунке 3.13). 136 операцию и те, что находятся выше нее по списку, нельзя выбрать (все они также заблокированы), поскольку она пересекается с предыдущими для 101 и 105 операций. Т.е. выбор операций в списке и проверки, происходящие в нем, осуществились в данном случае корректно.

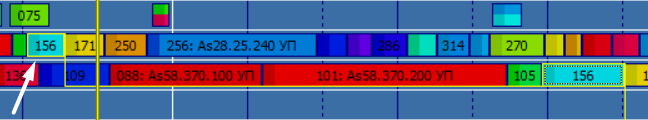


Рисунок 3.13. Причина, по которой 156 операция не проходит проверки.

После выбора необходимых операций в списке, кнопка «Объединить выбранные операции» в интерфейсе программной надстройки становится активной. Нажимаем ее, чтобы объединить операции с 109 по 105 включительно. После небольшой (5-секундной) задержки интерфейс программы обновляется, и можно увидеть объединенную операцию. Вид полученной объединенной операции изображен на рисунке 3.14. Между этой операцией и следующей на выбранном рабочем месте теперь можно увидеть участок свободного времени, которое появилось благодаря тому, что при объединении операций подготовительно-заключительное время не суммируется, а выбирается максимальным по всем операциям (т.к. подготовительно-заключительное время не зависит от объема операции). Теперь на освободившееся время возможно запланировать выполнение какой-то другой операции.

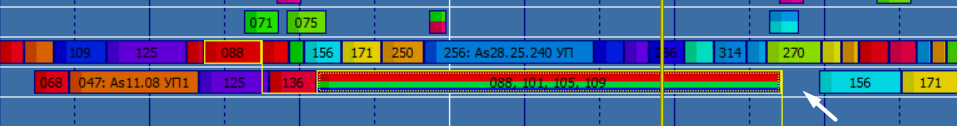


Рисунок 3.14. Объединенная операция, образованная одиночными операциями, и освободившееся рабочее время.

Список операций, которые можно объединить, также автоматически обновится в программной надстройке относительно объединенной операции, что показано на рисунке 3.145.

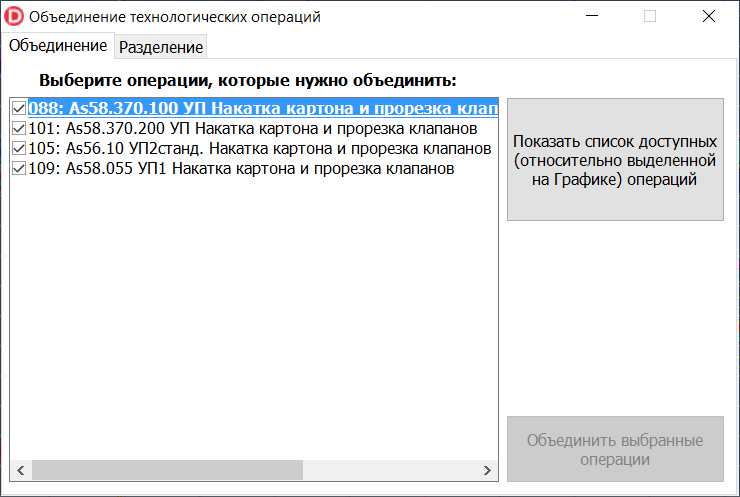


Рисунок 3.15. Вид интерфейса программы после объединения одиночных операций.

*2. Объединение одиночной операции с уже объединенной.*

Возьмем для примера 250, 256 одиночные операции и объединенную операцию 274,286,378. Исходное расположение операций на Графике загрузки рабочих мест показано на рисунке 3.16.

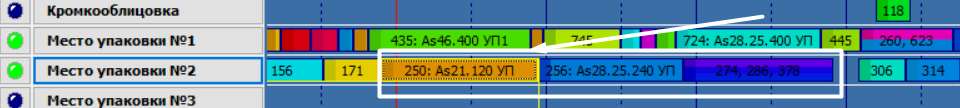


Рисунок 3.16. Одиночные операции 250, 256 и объединенная операция 274,286,378 на Графике загрузки рабочих мест.

Запустим программную надстройку для объединения технологических операций, выберем в системе 250 операцию и нажмем в программе на кнопку «Показать список доступных (относительно выделенной на Графике) операций». Выберем операции 250, 256, 274, 286 и 378. При этом при попытке выбрать операцию внутри объединенной (274 или 286) программа отменит выбор, не заблокировав все элементы объединения, поскольку в случае объединенных операций можно выбрать только всю такую операцию целиком, если все составляющие операции проходят проверки. Список в программе в итоге будет иметь вид, изображенный на рисунке 3.17.

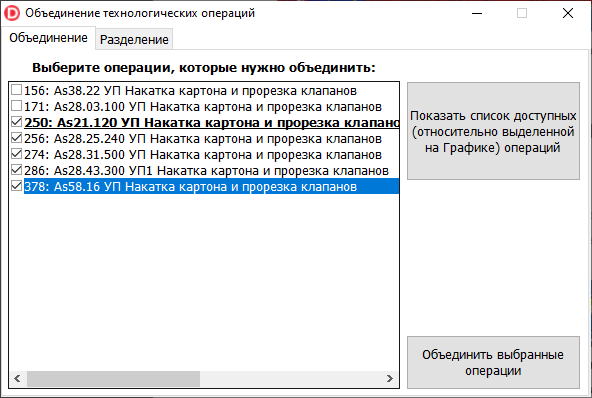


Рисунок 3.17. Выбор операций для объединения в Случае 2.

Далее нажимаем на ставшую активной кнопку «Объединить выбранные операции» и ожидаем примерно 5 секунд до обновления интерфейса программы. Объединенная операция, образованная одиночными операциями и объединением, показана на рисунке 3.18. Можно также увидеть после ее завершения на Графике освободившееся время.



Рисунок 3.18. Операция, образованная объединением одиночных операций и объединения и освободившееся рабочее время.

*3. Объединение объединенной операции с одиночной.*

Объединим объединенную операцию 270,365,374 и одиночные операции 378, 398 и 357. Изображение перечисленных операций в системе приведено на рисунке 3.19.

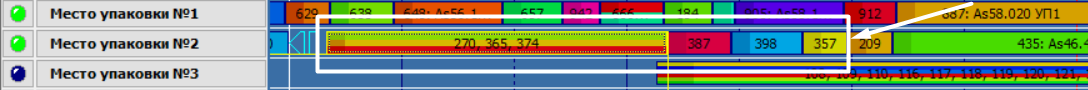


Рисунок 3.19. Объединенная операция 270,365,374 и одиночные операции 378, 398 и 357 на Графике загрузки рабочих мест.

Выберем в системе объединенную операцию 270,365,374 и нажмем в программной надстройке на кнопку «Показать список доступных (относительно выделенной на Графике) операций». Выберем остальные операции из перечисленных, как показано на рисунке 3.20. Можно заметить, что при попытке выбрать только часть из операций в объединении (270, 365 или 374), не выбрав при этом остальные, программа не позволит этого сделать, поскольку объединять можно только со всей объединенной операцией целиком.

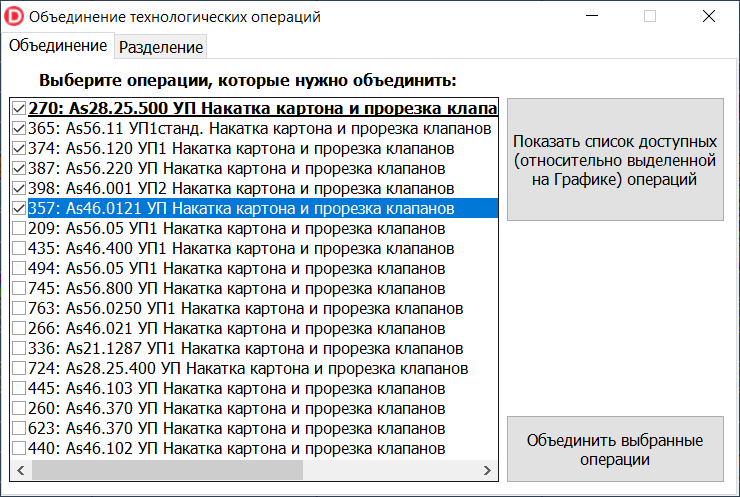


Рисунок 3.20. Выбор операций для объединения в Случае 3.

Нажимаем на кнопку «Объединить выбранные операции», снова ждем несколько секунд, пока интерфейс обновится. Полученная таким образом объединенная операция изображена на рисунке 3.21. Время после ее завершения на Графике также корректно высвободилось.

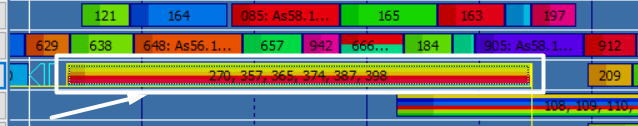


Рисунок 3.21. Объединенная операция, образованная объединением и одиночными операциями и освободившееся рабочее время.

*4. Объединение объединенных операций.*

Возьмем в качестве примера объединенные операции 270,357,365,374,387,398 и 209,435,494,745, приведенные на рисунке 3.22.



Рисунок 3.22. Две объединенных операции, идущие друг за другом на Графике загрузки рабочих мест.

Выделим первую операцию в системе и нажмем на кнопку «Показать список доступных (относительно выделенной на Графике) операций» в программной надстройке. В открывшемся списке выбираем также и вторую операцию. Можно заметить, что, если попытаться выбрать часть второй или первой объединенной операции, программа не даст этого сделать, но не заблокирует соответствующие ячейки. В итоге, список операций должен выглядеть так, как изображено на рисунке 3.23.

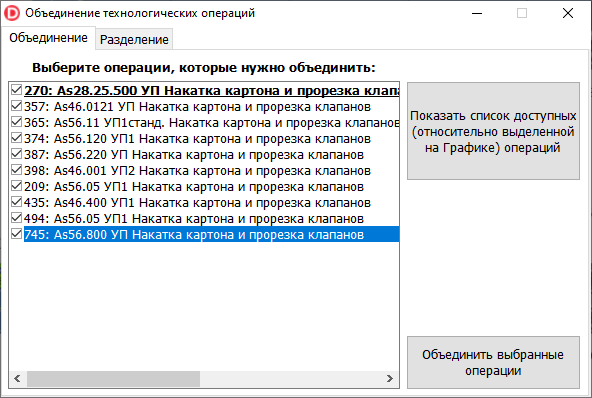


Рисунок 3.23. Выбор операций для объединения в Случае 4.

Нажимаем на разблокированную после выбора операций в списке кнопку «Объединить выбранные операции», ожидаем обновления интерфейса системы. Результирующая объединенная операция изображена на рисунке 3.24. Как и во всех остальных рассмотренных выше случаях, можно увидеть освободившееся в результате объединения операций рабочее время.



Рисунок 3.24. Объединенная операция, образованная из двух объединенных операций и освободившееся рабочее время.

Таким образом, во всех рассматриваемых случаях объединения операций программа отрабатывает корректно.

Далее рассмотрим разделение операций. Для того, чтобы осуществить разъединение операций, необходимо выбрать на Графике загрузки рабочих мест в системе объединенную операцию и переключиться на вкладку «Разделение» в программной надстройке «Объединение технологических операций». Выберем образованную объединением двух объединенных операций технологическую операцию (изображенную на рисунке 3.24 выше), перейдем на вкладку «Разделение» в программной надстройке и нажмем на кнопку «Проверить корректность разделения выбранной на Графике операции», чтобы увидеть список разделяемых операций. Полученный список изображен на рисунке 3.25.

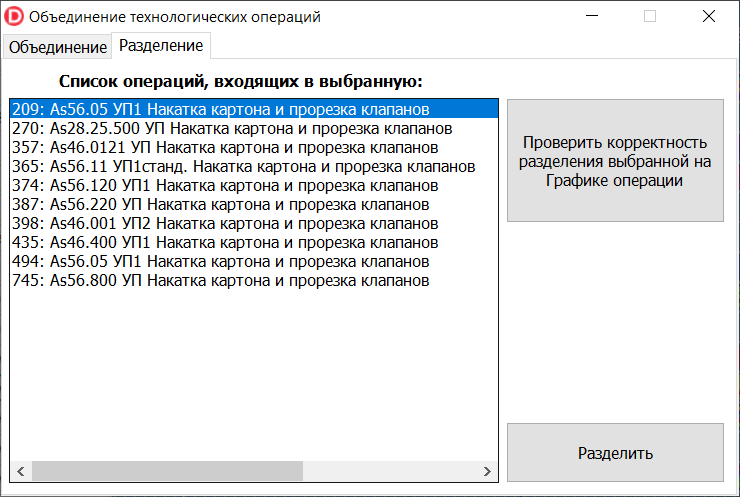


Рисунок 3.25. Список разделяемых операций.

После этого необходимо нажать на кнопку «Разделить» и подождать несколько секунд, чтобы интерфейс обновился. Операции вернутся к первоначальному (как было до объединения) состоянию, как видно из рисунка 3.26, время, освобожденное ранее объединением операций, снова будет занято.

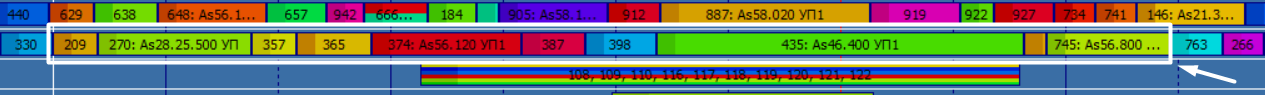


Рисунок 3.26. Результат разделения объединенной операции.

При этом список в программной надстройке станет пустым, поскольку выбранная операция больше не является объединенной, а кнопка «Разделить» снова будет заблокирована.

Если после объединения операций на освободившееся время будет запланирована какая-то другая операция, программа предупредит об этом и разделение операций произойти не сможет (до момента, пока не освободится снова достаточное количество времени).

Если попробовать разделить не объединенную операцию, программа также выдаст ошибку. Скриншот окна с ошибкой приведен на рисунке 3.27.

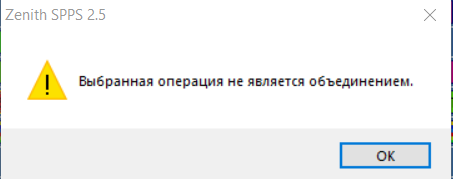


Рисунок 3.27. Окно с ошибкой при попытке разделения одиночной операции.

## 3.6 Вывод по главе

В ходе выполнения задач, поставленных во Введении данной выпускной квалификационной работы, была реализована программная надстройка на языке Delphi в виде DLL-файла, осуществляющая объединение однотипных технологических операций и их разъединение в случае такой необходимости, при условии выполнения вышеобозначенных ограничений и нюансов, а также проведено ее функциональное тестирование.

# Заключение

В ходе выполнения данной выпускной квалификационной работы была разработана программная надстройка MES-системы Zenith SPPS на языке программирования Delphi (Object Pascal) для объединения однотипных технологических операций при наличии возможности такого объединения.

Для достижения поставленной цели были решены следующие задачи:

1. Изучены особенности работы APS/MES-системы Zenith SPPS, непосредственно относящиеся к вышеобозначенной проблеме.

2. Составлены алгоритм объединения технологических операций, учитывающий все тонкости и нюансы, осуществляющий все необходимые проверки.

3. Продуман подходящий для решения задачи интерфейс.

4. Реализована программная надстройка, реализующая объединение и разъединение операций, в виде динамически загружаемой библиотеки на языке Delphi с целью интеграции ее как дополнительного модуля в APS/MES-систему Zenith SPPS.

5. Проведено функциональное тестирование полученного решения.

В ходе решения поставленных задач было реализовано программное обеспечение, позволяющее решить задачу объединения технологических операций в APS/MES-системе Zenith SPPS.

# Список литературы

1. Высочин, С.В. Руководство разработчика интерфейса Zenith SPPS система оперативного планирования Версия 2.5 [Текст] / С.В. Высочин. – Москва, 2020. – 203 с.

2. Голицына О. Л., Максимов Н. В., Попов И. И. Информационные системы. – 2014.

3. Валл Е. А. СИСТЕМЫ УПРАВЛЕНИЯ И ПЛАНИРОВАНИЯ ПРОИЗВОДСТВОМ //Аллея науки. – 2017. – Т. 2. – №. 11. – С. 208-211.

4. Лушников Н. Д., Альтерман А. Д. ИНТЕГРАЦИЯ MES-СИСТЕМЫ //Наука и образование: новое время. – 2019. – №. 1. – С. 73-76.

5. Савельева А. А., Варшавская В. В., Карлик Е. М. Использование MES системы в производственных процессах в условиях цифровизации //ТЕХНОЛОГИЧЕСКАЯ ПЕРСПЕКТИВА В РАМКАХ ЕВРАЗИЙСКОГО ПРОСТРАНСТВА: НОВЫЕ РЫНКИ И ТОЧКИ ЭКОНОМИЧЕСКОГО РОСТА. – 2019. – С. 283-286.

6. Амирханян А. Г., Амирханян Л. Г., Зобнина О. В. APS-системы как современный способ планирования производства //Modern Science. – 2020. – №. 6-3. – С. 36-39.

7. Официальный сайт APS/MES-системы Zenith SPPS [Электронный ресурс]. URL: http://www.zspps.ru

8. Высочин С., Смирнов Ю., Бояр-Созонович А. Внедрение MES-системы Zenith SPPS в различных производственных отраслях //САПР и графика – М.: Компьютер Пресс. – 2009. – №. 11. – С. 12-15.

9. Баурова Н. И., Зорин В. А., Приходько В. М. Технологическая наследственность и идентификация технологических процессов //Все материалы. Энциклопедический справочник. – 2015. – №. 2. – С. 2-7.

10. Шейкер, Т.Д. Разработка приложений баз данных в системе DELPHI [Текст]: учебное пособие / Т.Д. Шейкер. – Владивосток: Изд-во ДВГТУ – ДВГТУ, 2009. – 138 с.

11. Армстронг Т. ActiveX: Создание Web-приложений [Текст] / Т. Армстронг. – Киев: BHV – Санкт-Петербург, 1998. – 592 с.

# Приложение 1 Листинг разработанной программной надстройки для объединения технологических операций на языке программирования Delphi

Main.pas

unit Main;

interface

uses

  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

  System.Classes, Vcl.Graphics,

  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, Vcl.StdCtrls,

  System.Win.ComObj, Data.DB, Data.Win.ADODB, Vcl.ExtCtrls, System.Math,

  Vcl.CheckLst;

type

  TForm1 = class(TForm)

    DB: TADOConnection;

    Query: TADOQuery;

    PageControl1: TPageControl;

    FormSheet: TTabSheet;

    DeformSheet: TTabSheet;

    DeformShowButton: TButton;

    FormShowButton: TButton;

    FormListBox: TCheckListBox;

    FormLabel: TLabel;

    DeformLabel: TLabel;

    FormUnionButton: TButton;

    DeformDivideButton: TButton;

    DeformListBox: TListBox;

    procedure FormCreate(Sender: TObject);

    procedure FormShowButtonClick(Sender: TObject);

    procedure DeformShowButtonClick(Sender: TObject);

    procedure FormListBoxClickCheck(Sender: TObject);

    procedure FormListBoxDrawItem(Control: TWinControl; Index: Integer;

      Rect: TRect; State: TOwnerDrawState);

    procedure FormUnionButtonClick(Sender: TObject);

    procedure DeformDivideButtonClick(Sender: TObject);

  private

    { Private declarations }

  public

    { Public declarations }

  end;

var

  Form1: TForm1;

  ZenITh: OLEVariant;

  DB: TADOConnection;

  Query: TADOQuery;

  AppHWND: HWND;

  AppHandle: HWND;

  RemoteControl, IsAbort: boolean;

  CurTaskId, CurPrevFinish, CurNextStart, CurStart, CurFinish, SelMinIndex,

    SelMaxIndex: Integer;

  TaskIdList, PartIdList, CurItemIndex, ColorIdList, SelPartIdList,

    SelTaskIdList: array of Integer;

implementation

{$R \*.dfm}

procedure TForm1.FormCreate(Sender: TObject);

begin

  try

  begin

    // подключаемся к внутренней БД

    DB.ConnectionString := ZenITh.DBParameters;

    DB.Connected := true;

    ZenITh.ExecCommand(12);

  end

  except

    on E: Exception do

      raise Exception.Create('Невозможно подключиться к внутренней БД, ошибка: '

        + E.Message);

  end;

end;

// Возвращает строку, созданную из чисел массива через запятую (для IN в запросах)

function GetStringFromArray(var Arr: array of Integer): String;

var

  i: Integer;

  Str: String;

begin

  for i := 0 to High(Arr) do

  begin

    Str := Str + IntToStr(Arr[i]);

    if i <> High(Arr) then

      Str := Str + ',';

  end;

  Result := Str;

end;

procedure TForm1.FormShowButtonClick(Sender: TObject);

var

  i, j: Integer;

  // относительно текущей выбранной операции (в Зените)

  CurPlaceId, CurGroupId, Start, Finish, MinOperationLength: Integer;

  CurPartIdList: array of Integer;

  // относительно всех кандидатов в объединение

  IsAssembly, IsValidAssembly: Boolean;

  PartIdStrList, TaskIdStrList: String;

  minStartDateTime, maxFinishDateTime: TDateTime;

label

  l1, l2;

begin

  DeformListBox.Items.Clear;

  FormUnionButton.Enabled := false;

  CurStart := 0;

  CurFinish := 0;

  FormListBox.Items.Clear;

  // определить выбранную в Зените запись

  // номер операции

  CurTaskId := ZenITh.CurrentTaskID;

  if (CurTaskId = NULL) or (CurTaskId = 0) then

  begin

    ZenITh.ShowMessage

      ('Для корректной работы надстройки необходимо выбрать операцию на Графике загрузки рабочих мест.',

      '', mb\_IconStop);

    Exit;

  end;

  // номер рабочего места + вид работы + номера деталей + начало и конец выбранной операции

  // и проверка на то, чтобы GroupId совпадал

  with TADOQuery.Create(Self) do

  begin

    Connection := DB;

    SQL.Text :=

      'SELECT pt.PlaceId, pt.GroupId, pt.PartId, pt.Start, pt.Finish, pt.KindId, '

      + '(SELECT Finish FROM PlanTask p1 WHERE p1.PartId = pt.PartId AND p1.RecordPosition = pt.RecordPosition - 1) as PrevFinish, '

      + '(SELECT Start FROM PlanTask p2 WHERE p2.PartId = pt.PartId AND p2.RecordPosition = pt.RecordPosition + 1) as NextStart '

      + 'FROM PlanTask pt ' + 'WHERE pt.TaskId = ' + IntToStr(CurTaskId) + ' ';

    Open;

    First;

    i := 0;

    SetLength(CurPartIdList, RecordCount);

    CurGroupId := FieldByName('GroupId').AsInteger;

    CurPlaceId := FieldByName('PlaceId').AsInteger;

    CurStart := FieldByName('Start').AsInteger;

    CurFinish := FieldByName('Finish').AsInteger;

    CurPrevFinish := FieldByName('PrevFinish').AsInteger;

    CurNextStart := FieldByName('NextStart').AsInteger;

    while not eof do

    begin

      if FieldByName('KindId').AsInteger <> 10 then

      begin

        CurPartIdList := nil;

        FormListBox.Items.Clear;

        Close;

        Free;

        ZenITh.ShowMessage

          ('Выбранная операция не является технологической операцией.', '',

          mb\_IconExclamation);

        Exit;

      end;

      // в подходящем для объединения варианте GroupId совпадает (если сборка/объединение)

      if (i > 0) and (CurGroupId <> FieldByName('GroupId').AsInteger) then

      begin

        CurPartIdList := nil;

        FormListBox.Items.Clear;

        Close;

        Free;

        ZenITh.ShowMessage

          ('Операции в выбранной сборке относятся к разным видам работ, поэтому объединение невозможно.',

          '', mb\_IconStop);

        Exit;

      end;

      // ищем минимальное начало следующей операции и максимальное завершение предыдущей, чтобы отобрать соседние подходящие

      if CurPrevFinish < FieldByName('PrevFinish').AsInteger then

        CurPrevFinish := FieldByName('PrevFinish').AsInteger;

      if CurNextStart > FieldByName('NextStart').AsInteger then

        CurNextStart := FieldByName('NextStart').AsInteger;

      CurPartIdList[i] := FieldByName('PartId').AsInteger;

      Application.ProcessMessages;

      Next;

      inc(i);

    end;

    PartIdStrList := GetStringFromArray(PartIdList);

    SetLength(TaskIdList, 1);

    TaskIdList[0] := CurTaskId;

    // предыдущей и/или следующей может и не быть - тогда будут 0

    if CurPrevFinish = 0 then

      CurPrevFinish := -1;

    if CurNextStart = 0 then

      CurNextStart := -1;

    // минимальная длина операции в системе (константа)

    MinOperationLength := 0;

    SQL.Text :=

      'SELECT NumericValue FROM PlanConsts WHERE Name = ''MinOperationLength''';

    Open;

    First;

    MinOperationLength := FieldByName('NumericValue').AsInteger;

    // отбираем соседние операции, разрыв между ними должен быть меньше MinOperationLength, иначе поиск заканчиваем

    // сначала предыдущие

    IsAssembly := false;

    IsValidAssembly := true;

    Start := CurStart;

    Finish := CurFinish;

    while true do

    begin

      // +надо предусмотреть случаи, когда CurPrevFinish/CurNextStart = -1 - тогда нужно ограничение в пределах одного рабочего дня, иначе выдает безумное количество записей

      SQL.Text := 'SELECT pt.TaskId, pt.PartId, pt.KindId, ' +

        'pt.Start, pt.Finish, pt.GroupId, ' + 'pt.Assembly ' +

        'FROM PlanTask pt ' + 'WHERE pt.PlaceId = ' +

        IntToStr(CurPlaceId) + ' ' +

      // если это сборка, то у ее части может не совпадать GroupId - но нас может интересовать совпадающая часть

      // => соединять ТОЛЬКО по GroupId не выход

        'AND (pt.GroupId = ' + IntToStr(CurGroupId) + ' ' +

        'OR pt.Assembly = true) ' + 'AND pt.Finish <= ' + IntToStr(Start) +

        ' ' +

        'ORDER BY pt.Start DESC ';

      // от самой поздней к более ранним; поскольку нужна эта сортировка, придется делать в 2 прохода, тк поиск в 2 направлениях

      Open;

      if RecordCount = 0 then

        Break;

      First;

      while not eof do

      begin

        if (Start >= MinOperationLength + FieldByName('Finish').AsInteger)

        // разрыв между операциями больше минимальной длины операции в системе -> выход из всех вложенных циклов

          or (FieldByName('KindId').AsInteger <> 10)

        // или операция не является технологической операцией

          or ((CurPrevFinish <> -1) and (FieldByName('Start').AsInteger <

          CurPrevFinish))

        // операция должна начаться не раньше, чем завершится самая поздняя из предыдущих

        then

        // хотя использование goto не рекомендуется в общем случае, в случае выхода из вложенных циклов его использовать можно

        begin

          // добавить проверку на то, что это сборка и откатываться на counter элементов назад

          Close;

          Goto l1;

        end;

        // если эта операция является частью сборки/объединения, тогда проверить целиком на GroupId

        // "горизонтальный" поиск по таблице

        if FieldByName('Assembly').AsBoolean = true then

        begin

          // надо проверить всю сборку на совпадение по GroupId как-то и отобрать те части, где он совпадет

          // и нужен флаг, который укажет, что была сборка, но мы переключились на следующую после нее запись

          IsAssembly := true;

          if FieldByName('GroupId').AsInteger <> CurGroupId then

          begin

            // если не подходит, переключаемся на следующую запись и объявляем сборку невалидной

            IsValidAssembly := false;

            Next;

            continue;

          end;

        end;

        if (FieldByName('Assembly').AsBoolean = false) and (IsAssembly = true)

        then

        begin

          IsAssembly := false;

          // и на этом остановить поиск, если сборка невалидна, или продолжить его, если сборка валидна

          if IsValidAssembly = false then

          begin

            Close;

            Goto l1;

          end;

        end;

        // если все ок, сохраняем значения атрибутов

        // в случае более ранней (относительно выбранной) операции интересно только ее начало

        if Start > FieldByName('Start').AsInteger then

          Start := FieldByName('Start').AsInteger;

        // сохраняем TaskId

        SetLength(TaskIdList, Length(TaskIdList) + 1);

        TaskIdList[High(TaskIdList)] := FieldByName('TaskId').AsInteger;

        Next;

        Application.ProcessMessages;

      end;

    end;

    // затем последующие

  l1:

    IsAssembly := false;

    IsValidAssembly := true;

    while true do

    begin

      // сначала предыдущие

      SQL.Text := 'SELECT pt.TaskId, pt.PartId, pt.KindId, ' +

        'pt.Start, pt.Finish, pt.GroupId, ' + 'pt.Assembly ' +

        'FROM PlanTask pt ' + 'WHERE pt.PlaceId = ' + IntToStr(CurPlaceId) + ' '

        + 'AND (pt.GroupId = ' + IntToStr(CurGroupId) + ' ' +

        'OR pt.Assembly = true) ' + 'AND pt.Start >= ' + IntToStr(Finish) + ' '

        + 'ORDER BY pt.Start ASC ';

      // от самой ранней к более поздним

      Open;

      if RecordCount = 0 then

        Break;

      First;

      while not eof do

      begin

        if (Finish <= FieldByName('Start').AsInteger - MinOperationLength)

        // разрыв между операциями больше минимальной длины операции в системе -> выход из всех вложенных циклов

          or (FieldByName('KindId').AsInteger <> 10)

        // или операция не является технологической операцией

          or ((CurNextStart <> -1) and (FieldByName('Finish').AsInteger >

          CurNextStart))

        // операция должна завершиться не позже, чем начнется самая ранняя из последующих

        then

        // хотя использование goto не рекомендуется в общем случае, в случае выхода из вложенных циклов его использовать можно

        begin

          Close;

          Goto l2;

        end;

        // если эта операция является частью сборки/объединения, тогда проверить целиком на GroupId

        // "горизонтальный" поиск по таблице

        if FieldByName('Assembly').AsBoolean = true then

        begin

          // надо проверить всю сборку на совпадение по GroupId как-то и отобрать те части, где он совпадет

          // и нужен флаг, который укажет, что была сборка, но мы переключились на следующую после нее запись

          IsAssembly := true;

          if FieldByName('GroupId').AsInteger <> CurGroupId then

          begin

            // если не подходит, переключаемся на следующую запись и объявляем сборку невалидной

            IsValidAssembly := false;

            Next;

          end;

        end;

        if (FieldByName('Assembly').AsBoolean = false) and (IsAssembly = true)

        then

        begin

          IsAssembly := false;

          // и на этом остановить поиск

          if IsValidAssembly = false then

          begin

            Close;

            Goto l2;

          end;

        end;

        // если все ок, сохраняем значения атрибутов

        // в случае более поздней (относительно выбранной) операции интересен только ее конец

        if Finish < FieldByName('Finish').AsInteger then

          Finish := FieldByName('Finish').AsInteger;

        // сохраняем TaskId

        SetLength(TaskIdList, Length(TaskIdList) + 1);

        TaskIdList[High(TaskIdList)] := FieldByName('TaskId').AsInteger;

        Next;

        Application.ProcessMessages;

      end;

    end;

  l2:

    TaskIdStrList := GetStringFromArray(TaskIdList);

    SQL.Text :=

      'SELECT pp.RecordPosition, pp.PartName, pt.TaskName, pt.TaskId, pt.PartId, pp.Color '

      + 'FROM PlanTask pt ' + 'INNER JOIN PlanPart pp ' +

      'ON pt.PartId = pp.PartId ' + 'WHERE pt.TaskId IN (' + TaskIdStrList

      + ') ' + 'ORDER BY Start ASC, pp.RecordPosition ASC ';

    Open;

    First;

    i := 0;

    j := 0;

    SetLength(TaskIdList, 0);

    SetLength(PartIdList, 0);

    SetLength(CurItemIndex, 0);

    SetLength(ColorIdList, 0);

    while not eof do

    begin

      // нужно перестроить TaskIdList (отсортировать по Start), чтобы соответствие в FormListBox было при выборе элементов в нем

      SetLength(TaskIdList, Length(TaskIdList) + 1);

      SetLength(PartIdList, Length(PartIdList) + 1);

      SetLength(ColorIdList, Length(ColorIdList) + 1);

      TaskIdList[i] := FieldByName('TaskId').AsInteger;

      PartIdList[i] := FieldByName('PartId').AsInteger;

      ColorIdList[i] := FieldByName('Color').AsInteger;

      // и выводим его в общий список кандидатов для объединения

      if FieldByName('TaskId').AsInteger = CurTaskId then

      begin

        FormListBox.Items.Add(FormatFloat('000', FieldByName('RecordPosition')

          .AsInteger) + ': ' + FieldByName('PartName').AsString + ' ' +

          FieldByName('TaskName').AsString);

        SetLength(CurItemIndex, Length(CurItemIndex) + 1);

        CurItemIndex[j] := i;

        SelMaxIndex := i;

        inc(j);

        if j = 1 then

        begin

          FormListBox.Selected[i] := true;

          SelMinIndex := i;

        end;

        FormListBox.Checked[FormListBox.Items.Count - 1] := true;

        // надо, чтобы в самом начале выбранная операция была отмечена (или сборка вся)

      end

      else

      begin

        FormListBox.Items.Add(FormatFloat('000', FieldByName('RecordPosition')

          .AsInteger) + ': ' + FieldByName('PartName').AsString + ' ' +

          FieldByName('TaskName').AsString);

      end;

      Next;

      inc(i);

      Application.ProcessMessages;

    end;

    TaskIdStrList := GetStringFromArray(TaskIdList);

    Close;

    Free;

  end;

end;

procedure TForm1.FormListBoxDrawItem(Control: TWinControl; Index: Integer;

  // пока выделяется только 1 главная операция в сборке, а надо все

  Rect: TRect; State: TOwnerDrawState);

var

  i: Integer;

begin

  i := 0;

  with FormListBox.Canvas do

  begin

    if (Length(CurItemIndex) > 0) and (Index = CurItemIndex[i]) then

    begin

      Font.Style := [fsBold, fsUnderline];

      inc(i);

    end;

    // Font.Color := TColor(ColorIdList[Index]);

    TextRect(Rect, Rect.Left, Rect.Top, FormListBox.Items[Index]);

  end;

end;

// обработка выбора элементов в списках + проверки

procedure TForm1.FormListBoxClickCheck(Sender: TObject);

var

  i, j, CurMinIndex, CurMaxIndex, MaxPrevFinish, MinNextStart, SelStart,

    SelFinish, RangeFinish, Counter: Integer;

  IsValidUnion: Integer;

  // -2 проблема слева, -1 проблема справа, 0 проблемы и там и там, 1 - проблем нет

label

  s1, s2;

begin

  if PageControl1.ActivePage = FormSheet then

  begin

    SetLength(SelPartIdList, 0);

    SetLength(SelTaskIdList, 0);

    IsValidUnion := 1;

    // относительно выбранной в Зените операции

    CurMinIndex := MinIntValue(CurItemIndex);

    CurMaxIndex := MaxIntValue(CurItemIndex);

    MaxPrevFinish := CurPrevFinish;

    MinNextStart := CurNextStart;

    SelStart := CurStart;

    SelFinish := CurFinish;

    if FormListBox.ItemIndex < CurMinIndex then // выбраны слева от текущей

    begin

      // случай, когда  нажимают по одной и той же ячейке

      if SelMinIndex = FormListBox.ItemIndex then

        RangeFinish := FormListBox.ItemIndex + 1

      else

        RangeFinish := FormListBox.ItemIndex;

      with TADOQuery.Create(Self) do

      begin

        Connection := DB;

        Counter := 0;

        SetLength(SelPartIdList, Length(SelPartIdList) + 1);

        SetLength(SelTaskIdList, Length(SelTaskIdList) + 1);

        SelPartIdList[0] := PartIdList[SelMaxIndex];

        SelTaskIdList[0] := TaskIdList[SelMaxIndex];

        SelMinIndex := SelMaxIndex;

        for i := SelMaxIndex downto RangeFinish do

        // идем с противоположной границы до выделенного элемента

        begin

          // проверка по PartId

          if i < SelMaxIndex then

          begin

            SetLength(SelPartIdList, Length(SelPartIdList) + 1);

            SetLength(SelTaskIdList, Length(SelTaskIdList) + 1);

            SelPartIdList[High(SelPartIdList)] := PartIdList[i];

            SelTaskIdList[High(SelTaskIdList)] := TaskIdList[i];

            if SelTaskIdList[High(SelTaskIdList)] <> SelTaskIdList

              [High(SelTaskIdList) - 1] then

            begin

              Counter := 0;

            end;

            inc(Counter);

            for j := 0 to High(SelPartIdList) - 1 do

            begin

              if SelPartIdList[j] = PartIdList[i] then

              begin

                SelMinIndex := i + Counter;

                SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

                SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

                IsValidUnion := -2;

                Goto s1;

              end;

            end;

            if i = RangeFinish then

            begin

              if ((i > 0) and (SelTaskIdList[High(SelTaskIdList)] = TaskIdList

                [i - 1])) then

              begin

                SelMinIndex := i + Counter;

                SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

                SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

                Goto s1;

              end;

            end;

          end;

          // проверка пред/след

          SQL.Text :=

            'SELECT (SELECT p1.Finish FROM PlanTask p1 WHERE p1.PartId = pt.PartId AND p1.RecordPosition = pt.RecordPosition - 1) as PrevFinish, '

            +

          // предыдущая для текущей - важен только Finish - ее может и не быть, тогда все равно

            '(SELECT p2.Start FROM PlanTask p2 WHERE p2.PartId = pt.PartId AND p2.RecordPosition = pt.RecordPosition + 1) as NextStart, '

            +

          // следующая для текущей - важен только Start - ее может и не быть, тогда все равно

            'pt.Start, pt.Finish, pt.Assembly, pt.TaskName ' +

            'FROM PlanTask pt ' + 'WHERE pt.TaskId = ' +

            IntToStr(SelTaskIdList[High(SelTaskIdList)]) + ' ' +

          // только выделенные нужны уже

            'AND pt.PartId = ' +

            IntToStr(SelPartIdList[High(SelPartIdList)]) + ' ';

          Open;

          First;

          if ((MaxPrevFinish <> -1) and (FieldByName('Start').AsInteger <

            MaxPrevFinish)) or ((FieldByName('NextStart').AsInteger < SelFinish)

            or ((SelFinish > MinNextStart) and (MinNextStart <> -1))) then

          // не CurFinish в общем случае, а SelFinish

          begin

            SelMinIndex := i + Counter;

            FormListBox.Selected[SelMinIndex] := true;

            SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

            SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

            IsValidUnion := -2;

            Goto s1;

          end

          else // проверки пройдены

          begin

            // запоминаем значение переменных

            if FieldByName('PrevFinish').AsInteger <> 0 then

            begin

              if MaxPrevFinish = -1 then

                MaxPrevFinish := FieldByName('PrevFinish').AsInteger

              else if FieldByName('PrevFinish').AsInteger > MaxPrevFinish then

                MaxPrevFinish := FieldByName('PrevFinish').AsInteger;

            end;

            if FieldByName('NextStart').AsInteger <> 0 then

            begin

              if MinNextStart = -1 then

                MinNextStart := FieldByName('NextStart').AsInteger

              else if FieldByName('NextStart').AsInteger < MinNextStart then

                MinNextStart := FieldByName('NextStart').AsInteger;

            end;

            if FieldByName('Start').AsInteger < SelStart then

              SelStart := FieldByName('Start').AsInteger;

            if FieldByName('Finish').AsInteger > SelFinish then

              SelFinish := FieldByName('Finish').AsInteger;

            SelMinIndex := i;

          end;

          Close;

        end;

        Free;

      end;

    end

    else if (FormListBox.ItemIndex >= CurMinIndex) and

      (FormListBox.ItemIndex <= CurMaxIndex) then // выбраны текущие

    begin

      SelMinIndex := CurMinIndex;

      SelMaxIndex := CurMaxIndex;

      MaxPrevFinish := CurPrevFinish;

      MinNextStart := CurNextStart;

      SelStart := CurStart;

      SelFinish := CurFinish;

      FormUnionButton.Enabled := false;

      IsValidUnion := 1;

    end

    else if FormListBox.ItemIndex > CurMaxIndex then

    // выбраны справа от текущей

    begin

      if SelMaxIndex = FormListBox.ItemIndex then

        RangeFinish := FormListBox.ItemIndex - 1

      else

        RangeFinish := FormListBox.ItemIndex;

      with TADOQuery.Create(Self) do

      begin

        Connection := DB;

        Counter := 0;

        SetLength(SelPartIdList, Length(SelPartIdList) + 1);

        SetLength(SelTaskIdList, Length(SelTaskIdList) + 1);

        SelPartIdList[0] := PartIdList[SelMinIndex];

        SelTaskIdList[0] := TaskIdList[SelMinIndex];

        SelMaxIndex := SelMinIndex;

        for i := SelMinIndex to RangeFinish do

        // идем с противоположной границы до выделенного элемента

        begin

          // проверка по PartId

          if i > SelMinIndex then

          begin

            SetLength(SelPartIdList, Length(SelPartIdList) + 1);

            SetLength(SelTaskIdList, Length(SelTaskIdList) + 1);

            SelPartIdList[High(SelPartIdList)] := PartIdList[i];

            SelTaskIdList[High(SelTaskIdList)] := TaskIdList[i];

            if SelTaskIdList[High(SelTaskIdList)] <> SelTaskIdList

              [High(SelTaskIdList) - 1] then

            begin

              Counter := 0;

            end;

            inc(Counter);

            for j := 0 to High(SelPartIdList) - 1 do

            begin

              if SelPartIdList[j] = PartIdList[i] then

              begin

                SelMaxIndex := i - Counter;

                SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

                SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

                IsValidUnion := -1;

                Goto s1;

              end;

            end;

            if i = RangeFinish then

            begin

              if ((i < High(TaskIdList)) and (SelTaskIdList[High(SelTaskIdList)

                ] = TaskIdList[i + 1])) then

              begin

                SelMaxIndex := i - Counter;

                SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

                SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

                Goto s1;

              end;

            end;

          end;

          // проверка пред/след

          SQL.Text :=

            'SELECT (SELECT p1.Finish FROM PlanTask p1 WHERE p1.PartId = pt.PartId AND p1.RecordPosition = pt.RecordPosition - 1) as PrevFinish, '

            +

          // предыдущая для текущей - важен только Finish - ее может и не быть, тогда все равно

            '(SELECT p2.Start FROM PlanTask p2 WHERE p2.PartId = pt.PartId AND p2.RecordPosition = pt.RecordPosition + 1) as NextStart, '

            +

          // следующая для текущей - важен только Start - ее может и не быть, тогда все равно

            'pt.Start, pt.Finish ' + 'FROM PlanTask pt ' + 'WHERE pt.TaskId = '

            + IntToStr(SelTaskIdList[High(SelTaskIdList)]) + ' ' +

          // только выделенные нужны уже

            'AND pt.PartId = ' +

            IntToStr(SelPartIdList[High(SelPartIdList)]) + ' ';

          Open;

          First;

          if ((MinNextStart <> -1) and (FieldByName('Finish').AsInteger >

            MinNextStart)) or ((FieldByName('PrevFinish').AsInteger > SelStart)

            or (SelStart < MaxPrevFinish)) then

          // не CurStart в общем случае, а SelStart, тк могли быть выделения ранее

          begin

            SelMaxIndex := i - Counter;

            FormListBox.Selected[SelMaxIndex] := true;

            SetLength(SelPartIdList, Length(SelPartIdList) - Counter);

            SetLength(SelTaskIdList, Length(SelTaskIdList) - Counter);

            IsValidUnion := -1;

            Goto s1;

          end

          else // проверки пройдены

          begin

            // запоминаем значение переменных

            if FieldByName('PrevFinish').AsInteger <> 0 then

            begin

              if MaxPrevFinish = -1 then

                MaxPrevFinish := FieldByName('PrevFinish').AsInteger

              else if FieldByName('PrevFinish').AsInteger > MaxPrevFinish then

                MaxPrevFinish := FieldByName('PrevFinish').AsInteger;

            end;

            if FieldByName('NextStart').AsInteger <> 0 then

            begin

              if MinNextStart = -1 then

                MinNextStart := FieldByName('NextStart').AsInteger

              else if FieldByName('NextStart').AsInteger < MinNextStart then

                MinNextStart := FieldByName('NextStart').AsInteger;

            end;

            if FieldByName('Start').AsInteger < SelStart then

              SelStart := FieldByName('Start').AsInteger;

            if FieldByName('Finish').AsInteger > SelFinish then

              SelFinish := FieldByName('Finish').AsInteger;

            SelMaxIndex := i;

          end;

          Close;

        end;

        Free;

      end;

    end;

  s1:

    // сами выделения и блокировки

    // придется идти в 2 прохода

    with TADOQuery.Create(Self) do

    begin

      Connection := DB;

      // сначала слева

      for i := SelMinIndex - 1 downto 0 do

      begin

        FormListBox.Checked[i] := false;

        FormListBox.Refresh;

        if (IsValidUnion = -2) or (IsValidUnion = 0) then

        begin

          FormListBox.ItemEnabled[i] := false;

          FormListBox.Refresh;

        end

        else

        begin

          SQL.Text := 'SELECT pt.Start ' + 'FROM PlanTask pt ' +

            'WHERE pt.TaskId = ' + IntToStr(TaskIdList[i]) + ' ' +

            'AND pt.PartId = ' + IntToStr(PartIdList[i]) + ' ';

          Open;

          First;

          if FieldByName('Start').AsInteger < MaxPrevFinish then

          begin

            FormListBox.ItemEnabled[i] := false;

            FormListBox.Refresh;

            if IsValidUnion = -1 then

              IsValidUnion := 0

            else

              IsValidUnion := -2;

          end

          else

          begin

            FormListBox.ItemEnabled[i] := true;

            FormListBox.Refresh;

          end;

          Close;

        end;

        Application.ProcessMessages;

      end;

      // затем выделенные

      for i := SelMinIndex to SelMaxIndex do

      begin

        FormListBox.Checked[i] := true;

        FormListBox.ItemEnabled[i] := true;

        Application.ProcessMessages;

        FormListBox.Refresh;

      end;

      // затем те, что справа

      for i := SelMaxIndex + 1 to High(PartIdList) do

      begin

        FormListBox.Checked[i] := false;

        FormListBox.Refresh;

        if (IsValidUnion = -1) or (IsValidUnion = 0) then

        begin

          FormListBox.ItemEnabled[i] := false;

          FormListBox.Refresh;

        end

        else

        begin

          SQL.Text := 'SELECT pt.Finish ' + 'FROM PlanTask pt ' +

            'WHERE pt.TaskId = ' + IntToStr(TaskIdList[i]) + ' ' +

            'AND pt.PartId = ' + IntToStr(PartIdList[i]) + ' ';

          Open;

          First;

          if FieldByName('Finish').AsInteger > MinNextStart then

          begin

            FormListBox.ItemEnabled[i] := false;

            FormListBox.Refresh;

            IsValidUnion := -1;

          end

          else

          begin

            FormListBox.ItemEnabled[i] := true;

            FormListBox.Refresh;

          end;

          Close;

        end;

        Application.ProcessMessages;

      end;

      Free;

      if (SelMinIndex = CurMinIndex) and (SelMaxIndex = CurMaxIndex) then

        FormUnionButton.Enabled := false

      else

        FormUnionButton.Enabled := true;

    end;

  end;

end;

// объединение операций

procedure TForm1.FormUnionButtonClick(Sender: TObject);

var

  i, MinStart, MaxFinish, SumProcessing, MinTaskId, MaxPreparation, SelTaskId: Integer;

  minStartDateTime, maxFinishDateTime: TDateTime;

  SelTaskIdStrList, SelPartIdStrList: String;

begin

  // поместить выделенные в списке операции в строку для запросов с диапазоном

  SelTaskIdStrList := GetStringFromArray(SelTaskIdList);

  SelPartIdStrList := GetStringFromArray(SelPartIdList);

  with TADOQuery.Create(Self) do

  begin

    Connection := DB;

    // чтобы после объединения в системе объединенная операция осталась выбранной

    SQL.Text := 'SELECT pt.TaskId AS SelTaskId ' +

                'FROM PlanTask pt ' +

                'INNER JOIN PlanPart pp ' +

                'ON pt.PartId = pp.PartId ' +

                'WHERE pp.RecordPosition = (SELECT MIN(pp1.RecordPosition) ' +

                'FROM PlanPart pp1 WHERE pp1.PartId = pp.PartId)' +

                'AND pt.TaskId IN (' + SelTaskIdStrList + ') ' +

                'AND pt.PartId IN (' + SelPartIdStrList + ') ';

    Open;

    First;

    SelTaskId := FieldByName('SelTaskId').AsInteger;

    SQL.Text :=

      'SELECT (SUM(p.Processing) + MAX(p.Preparation)) AS CalcProcessing, ' +

      'MIN(p.Start) AS MinStart, ' +

      'MIN(p.StartDateTime) AS MinStartDateTime, ' +

      'MIN(p.MTaskId) AS MinTaskId, ' + 'MAX(p.Preparation) AS MaxPreparation '

      + 'FROM (SELECT MIN(pt.TaskId) as MTaskId, ' +

      'MIN(pt.Processing - pt.Preparation) as Processing, ' +

      'MAX(pt.Preparation) as Preparation, ' + 'MIN(pt.TaskId) as TaskId, ' +

      'MIN(pt.Start) as Start, ' + 'MIN(pt.StartDateTime) as StartDateTime ' +

      'FROM PlanTask pt ' +

      'WHERE pt.TaskId IN (' + SelTaskIdStrList + ') ' + 'AND pt.PartId IN(' +

      SelPartIdStrList + ') ' + 'GROUP BY pt.TaskId) AS p ';

    Open;

    First; // запись будет 1, тк агрегации

    MinTaskId := FieldByName('MinTaskId').AsInteger;

    MinStart := FieldByName('MinStart').AsInteger;

    minStartDateTime := FieldByName('MinStartDateTime').AsDateTime;

    SumProcessing := FieldByName('CalcProcessing').AsInteger;

    MaxPreparation := FieldByName('MaxPreparation').AsInteger;

    Close;

    MaxFinish := MinStart + SumProcessing;

    maxFinishDateTime := ZenITh.PosToTime[MaxFinish];

    try

      DB.BeginTrans;

      ParamCheck := true;

      SQL.Text := 'UPDATE PlanTask ' + 'SET TaskId = :p1, ' + 'Start = :p2, ' +

        'Finish = :p3, ' + 'Processing = :p4, ' + 'StartDateTime = :p5, ' +

        'FinishDateTime = :p6, ' + 'Preparation = :p7, ' + 'Assembly = true ' +

        'WHERE TaskId IN(' + SelTaskIdStrList + ') ' + 'AND PartId IN( ' +

        SelPartIdStrList + ') ';

      Parameters.ParamByName('p1').Value := MinTaskId;

      Parameters.ParamByName('p2').Value := MinStart;

      Parameters.ParamByName('p3').Value := MaxFinish;

      Parameters.ParamByName('p4').Value := SumProcessing;

      Parameters.ParamByName('p5').Value := minStartDateTime;

      Parameters.ParamByName('p6').Value := maxFinishDateTime;

      Parameters.ParamByName('p7').Value := MaxPreparation;

      ExecSQL();

      DB.CommitTrans;

    except

      on E: Exception do

      begin

        DB.RollbackTrans;

        ZenITh.ShowMessage

          ('Не удалось внести данные в базу данных. Попробуйте повторить операцию',

          '', mb\_IconStop);

        Close;

        Free;

        Exit;

      end;

    end;

    // обновить данные в приложении

    // задержка на 5 секунд перед обновлением

    Screen.Cursor := crAppStart;

    Sleep(5000);

    // обновление данных в основной программе

    ZenITh.CurrentTaskID := SelTaskId;

    ZenITh.Refresh;

    Form1.FormShowButtonClick(FormShowButton);

    Screen.Cursor := crDefault;

    Close;

    Free;

    FormUnionButton.Enabled := false;

  end;

end;

procedure TForm1.DeformShowButtonClick(Sender: TObject);

var

  CurPlaceId, FreeTime, SumPreparation, Preparation: Integer;

begin

  DeformDivideButton.Enabled := false;

  DeformListBox.Items.Clear;

  FormListBox.Items.Clear;

  SetLength(TaskIdList, 0);

  SetLength(PartIdList, 0);

  CurTaskId := ZenITh.CurrentTaskID;

  if (CurTaskId = NULL) or (CurTaskId = 0) then

  begin

    ZenITh.ShowMessage

      ('Для корректной работы надстройки необходимо выбрать операцию на Графике загрузки рабочих мест.',

      '', mb\_IconStop);

    Exit;

  end;

  SumPreparation := 0;

  Preparation := 0;

  with TADOQuery.Create(Self) do

  begin

    Connection := DB;

    SQL.Text :=

      'SELECT pp.RecordPosition, pp.PartName, pt.TaskName, pt.Assembly, ' +

      'pt.TaskId, pt.PartId, pt.PlaceId, pt.Preparation, pt.Finish, pt.Start ' +

      'FROM PlanTask pt ' + 'INNER JOIN PlanPart pp ' +

      'ON pt.PartId = pp.PartId ' + 'WHERE pt.TaskId = ' + IntToStr(CurTaskId) +

      ' ';

    Open;

    First;

    while not eof do

    begin

      if FieldByName('TaskName').AsString = '\*' then

      begin

        DeformListBox.Clear;

        SetLength(TaskIdList, 0);

        SetLength(PartIdList, 0);

        ZenITh.ShowMessage

          ('Выбранную операцию нельзя разделить, поскольку она является сборочной.',

          '', mb\_IconExclamation);

        // иначе не понятно, как для таких операций восстанавливать TaskName

        Exit;

      end

      else if FieldByName('Assembly').AsBoolean = false then

      begin

        DeformListBox.Clear;

        SetLength(TaskIdList, 0);

        SetLength(PartIdList, 0);

        ZenITh.ShowMessage('Выбранная операция не является объединением.', '',

          mb\_IconExclamation);

        Exit;

      end

      else

      begin

        DeformListBox.Items.Add(FormatFloat('000', FieldByName('RecordPosition')

          .AsInteger) + ': ' + FieldByName('PartName').AsString + ' ' +

          FieldByName('TaskName').AsString);

        SetLength(TaskIdList, Length(TaskIdList) + 1);

        SetLength(PartIdList, Length(PartIdList) + 1);

        TaskIdList[High(TaskIdList)] := FieldByName('TaskId').AsInteger;

        PartIdList[High(PartIdList)] := FieldByName('PartId').AsInteger;

        CurPlaceId := FieldByName('PlaceId').AsInteger;

        CurStart := FieldByName('Start').AsInteger;

        CurFinish := FieldByName('Finish').AsInteger;

        Preparation := FieldByName('Preparation').AsInteger;

        SumPreparation := SumPreparation + FieldByName('Preparation').AsInteger;

        Application.ProcessMessages;

        Next;

      end;

    end;

    // проверка, что времени до следующих операций хватит на то, чтобы разделить выбранную

    SQL.Text := 'SELECT TOP 1 pt.Start ' + 'FROM PlanTask pt ' +

      'WHERE pt.PlaceId = ' + IntToStr(CurPlaceId) + ' ' + 'AND pt.Start >= ' +

      IntToStr(CurFinish) + ' ' + 'ORDER BY pt.Start ASC';

    Open;

    First;

    FreeTime := FieldByName('Start').AsInteger - CurFinish;

    if FreeTime < (SumPreparation - Preparation) then

    begin

      DeformListBox.Clear;

      SetLength(TaskIdList, 0);

      SetLength(PartIdList, 0);

      ZenITh.ShowMessage

        ('Разделение выбранной операции невозможно, поскольку не достаточно времени до следующей операции.',

        '', mb\_IconExclamation);

      Exit;

    end;

    // если же все нормально, тогда можно разделять операцию

    DeformDivideButton.Enabled := true;

    Close;

    Free;

  end;

end;

procedure TForm1.DeformDivideButtonClick(Sender: TObject);

var

  NewTaskId, i, Start, Finish, SelPartId, SelTaskId: Integer;

  ProcessingList, PreparationList: array of Integer;

begin

  SetLength(ProcessingList, 0);

  SetLength(PreparationList, 0);

  with TADOQuery.Create(Self) do

  begin

    Connection := DB;

    // рассчитаем необходимые параметры - Processing и TaskId

    try

      begin

        // чтобы после объединения в системе объединенная операция осталась выбранной

        SQL.Text := 'SELECT pt.PartId AS SelPartId ' +

                    'FROM PlanTask pt ' +

                    'INNER JOIN PlanPart pp ' +

                    'ON pt.PartId = pp.PartId ' +

                    'WHERE pp.RecordPosition = (SELECT MIN(pp1.RecordPosition) ' +

                    'FROM PlanPart pp1 WHERE pp1.PartId = pp.PartId)' +

                    'AND pt.TaskId = ' + IntToStr(CurTaskId) + ' ';

        Open;

        First;

        SelPartId := FieldByName('SelPartId').AsInteger;

        // получаем новый TaskId

        NewTaskId := ZenITh.NewIDValue[208];

        for i := 0 to High(PartIdList) do

        begin

          SQL.Text :=

            'SELECT (ROUND(SWITCH(pg.portion = -1, IIF(pt.TaskVolume IS NULL, '

            + 'pp.PartVolume, pt.TaskVolume) \* IIF(pt.InitProcessing IS NULL, '

            + 'pg.InitProcessing, pt.InitProcessing), ' +

            'pg.portion = 0, IIF(pt.InitProcessing IS NULL, pg.InitProcessing, pt.InitProcessing), '

            + 'pg.portion > 0,  (IIF(pt.TaskVolume IS NULL, pp.PartVolume, pt.TaskVolume) \* '

            + 'IIF(pt.InitProcessing IS NULL, pg.InitProcessing, pt.InitProcessing))/pg.Portion)) '

            + '+ IIF(pt.InitPreparation IS NULL, pg.InitPreparation, pt.InitPreparation)) AS CalcProcessing, '

            + 'IIF(pt.InitPreparation IS NULL, pg.InitPreparation, pt.InitPreparation) AS CalcPreparation '

            + 'FROM  (PlanTask pt ' + 'INNER JOIN PlanPart pp ' +

            'ON (pt.PartId = pp.PartId)) ' + 'INNER JOIN PlanGrpl pg ' +

            'ON ((pt.GroupId = pg.GroupId) AND (pt.PlaceId = pg.PlaceId)) ' +

            'WHERE pt.TaskId = ' + IntToStr(CurTaskId) + ' ' +

            'AND pt.PartId = ' + IntToStr(PartIdList[i]) + ' ';

          Open;

          First;

          while not eof do

          begin

            SetLength(ProcessingList, Length(ProcessingList) + 1);

            SetLength(PreparationList, Length(PreparationList) + 1);

            ProcessingList[i] := FieldByName('CalcProcessing').AsInteger;

            PreparationList[i] := FieldByName('CalcPreparation').AsInteger;

            Next;

          end;

        end;

        for i := 0 to High(PartIdList) do

        begin

          if i = 0 then

            Start := CurStart

          else

            Start := Start + ProcessingList[i - 1];

          Finish := Start + ProcessingList[i];

          DB.BeginTrans;

          ParamCheck := true;

          if PartIdList[i] = SelPartId then

            SelTaskId := NewTaskId + i;

          SQL.Text := 'UPDATE PlanTask ' + 'SET TaskId = ' +

            IntToStr(NewTaskId + i) + ', ' + 'Assembly = false, ' +

            'Processing = ' + IntToStr(ProcessingList[i]) + ', ' +

            'Preparation = ' + IntToStr(PreparationList[i]) + ', ' + 'Start = '

            + IntToStr(Start) + ', ' + 'Finish = ' + IntToStr(Finish) + ', ' +

            'StartDateTime = :p1, ' + 'FinishDateTime = :p2 ' +

            'WHERE TaskId = ' + IntToStr(CurTaskId) + ' ' + 'AND PartId = ' +

            IntToStr(PartIdList[i]) + ' ';

          Parameters.ParamByName('p1').Value := ZenITh.PosToTime[Start];

          Parameters.ParamByName('p2').Value := ZenITh.PosToTime[Finish];

          ExecSQL;

          DB.CommitTrans;

          DeformListBox.Clear;

          DeformDivideButton.Enabled := false;

        end;

      end;

    except

      begin

        DB.RollbackTrans;

        Form1.DeformShowButtonClick(DeformShowButton);

        ZenITh.ShowMessage

          ('Не удалось внести данные в базу данных. Попробуйте повторить операцию',

          '', mb\_IconStop);

      end;

    end;

    // обновить данные в приложении

    // задержка на 5 секунд перед обновлением

    Screen.Cursor := crAppStart;

    Sleep(5000);

    // обновление данных в основной программе

    ZenITh.CurrentTaskID := SelTaskId;

    ZenITh.Refresh;

    Screen.Cursor := crDefault;

    Close;

    Free;

  end;

end;

end.

CompTechOperations.dpr

library CombTechOperations;

uses

  System.SysUtils,

  Classes,

  Windows,

  Vcl.Forms,

  Vcl.Controls,

  ComObj,

  Variants,

  ADODB,

  Main in 'Main.pas' {Form1};

{$R \*.res}

procedure AfterLoad(AppHandle: HWND); stdcall;

begin

  try

  Application.Initialize;

    try

      ZenITh := CreateOleObject('ZenPlan.ZenApp');

    except

      Application.MessageBox('Невозможно запустить сервер Zenith SPPS.' +

                             'Для регистрации OLE-сервера запустите ' +

                             'файл ZenPlan.exe c параметром /regserver',

                             'Действие невозможно!', mb\_IconStop);

      Exit; {!!}

    end;

    AppHWND := Application.Handle;

    Application.MainFormOnTaskbar := True;

    Application.CreateForm(TForm1, Form1);

  Application.Run;

  finally

    Application.Handle := AppHWND;

    ZenITh := Unassigned;

  end;

end;

exports

AfterLoad;

begin

end.