УДК 004.942:004.4'244

СПОСОБЫ ИНТЕГРАЦИИ ИМИТАЦИОННЫХ МОДЕЛЕЙ В ИНФОРМАЦИОННУЮ СИСТЕМУ ЗАКАЗЧИКА

В.Д. Левчук, П.Л. Чечет

Гомельский государственный университет им. Ф. Скорины, Гомель

THE TECHNICS OF SIMULATION MODELS IMPLEMENTATION INTO THE INFORMATION SYSTEM OF COMPANY

V.D. Liauchuk, P.L. Chechat

F. Scorina Gomel State University, Gomel

Рассматриваются вопросы интеграции имитационных моделей в информационную среду заказчика. На примерах разработанных имитационных моделей рассмотрено использование различных контейнеров ввода/вывода и модулей преобразования форматов результатов моделирования.

Ключевые слова: *имитационная модель, контейнеры, информационная среда, форматы обмена данными.*

The questions of simulation models integration into the customers information environment is described. Simulation models are used as an example of the use and development of different input/output containers. The format of the output results conversion is also described.

Keywords: *simulation model, containers, information environment, data exchange formats.*

Введение

В большинстве случаев заказчик имитационной модели (ИМ) планирует её использование совместно с теми программными продуктами, которые он использует в своей обычной (производственной, обслуживающей, учётной и др.) деятельности. Как минимум, исходные данные для имитационной модели будут браться из некоторого программного обеспечения, используемого заказчиком. Удобство работы с имитационной моделью во многом зависит от того, насколько автоматизированы будут подготовка исходных данных для моделирования и перенос результатов моделирования в привычное для заказчика программное обеспечение. Общее впечатление персонала и эффективность использования имитационной модели во многом определяются удобством её использования, поэтому этим вопросам следует уделять при разработке имитационной модели повышенное внимание.

1 Возможные реализации и работа с программой имитационной модели

Реализация программы ИМ в виде программного модуля на некотором языке программирования позволяет использовать различные контейнеры для ее выполнения: консольное приложение, WEB-приложение, приложение с графическим интерфейсом пользователя, АСУ предприятия и др.

Пример реализации программы ИМ с использованием в качестве контейнера консольного приложения, приведен на рисунке 1.

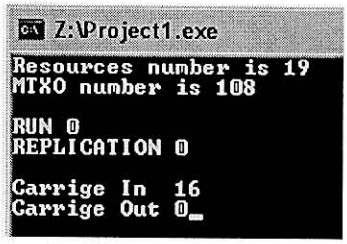


Рисунок 1 – Консольная программа ИМ

Структура текстовых файлов, создаваемых такой моделью, может быть адаптирована для просмотра пользователем или для непосредственного открытия в требуемых программах статистической обработки данных. Пример открытия файла входных параметров, адаптированного для открытия в табличном процессоре Microsoft Excel приведен на рисунке 2.

Вторым примером возможной реализации контейнера программы ИМ является Win32 программа с графическим пользовательским интерфейсом (рисунок 3). В таком случае пользователь получает оконное приложение, в элементы управления которого заносятся значения параметров и переменных ИМ. Нажатие кнопки «Запуск» приводит к запуску процесса моделирования, по окончании которого результаты помещаются на закладке «Отклики». Использование такого контейнера позволяет значительно упростить интерактивное взаимодействие пользователя с ИМ, позволяя оперативно вручную изменять входные данные и анализировать поведение ИМ.

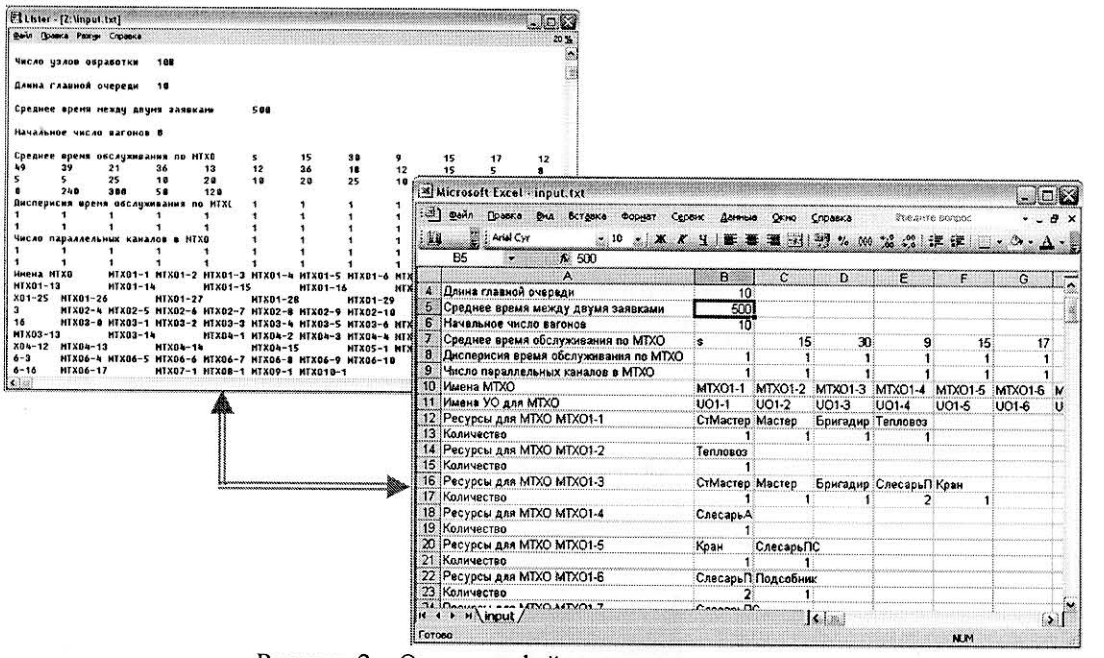


Рисунок 2 – Открытие файла входных параметров

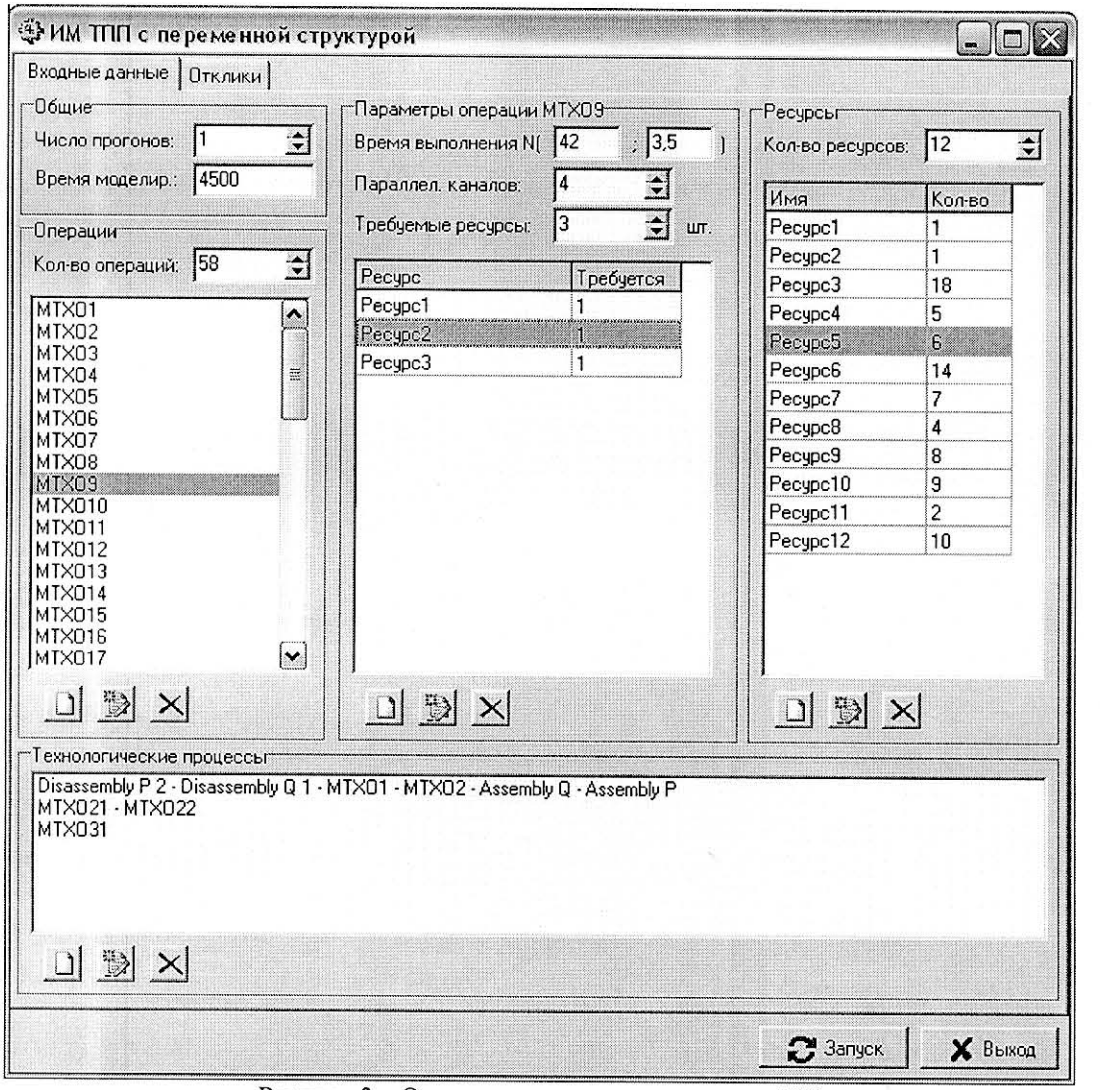


Рисунок 3 – Оконная реализация программы ИМ

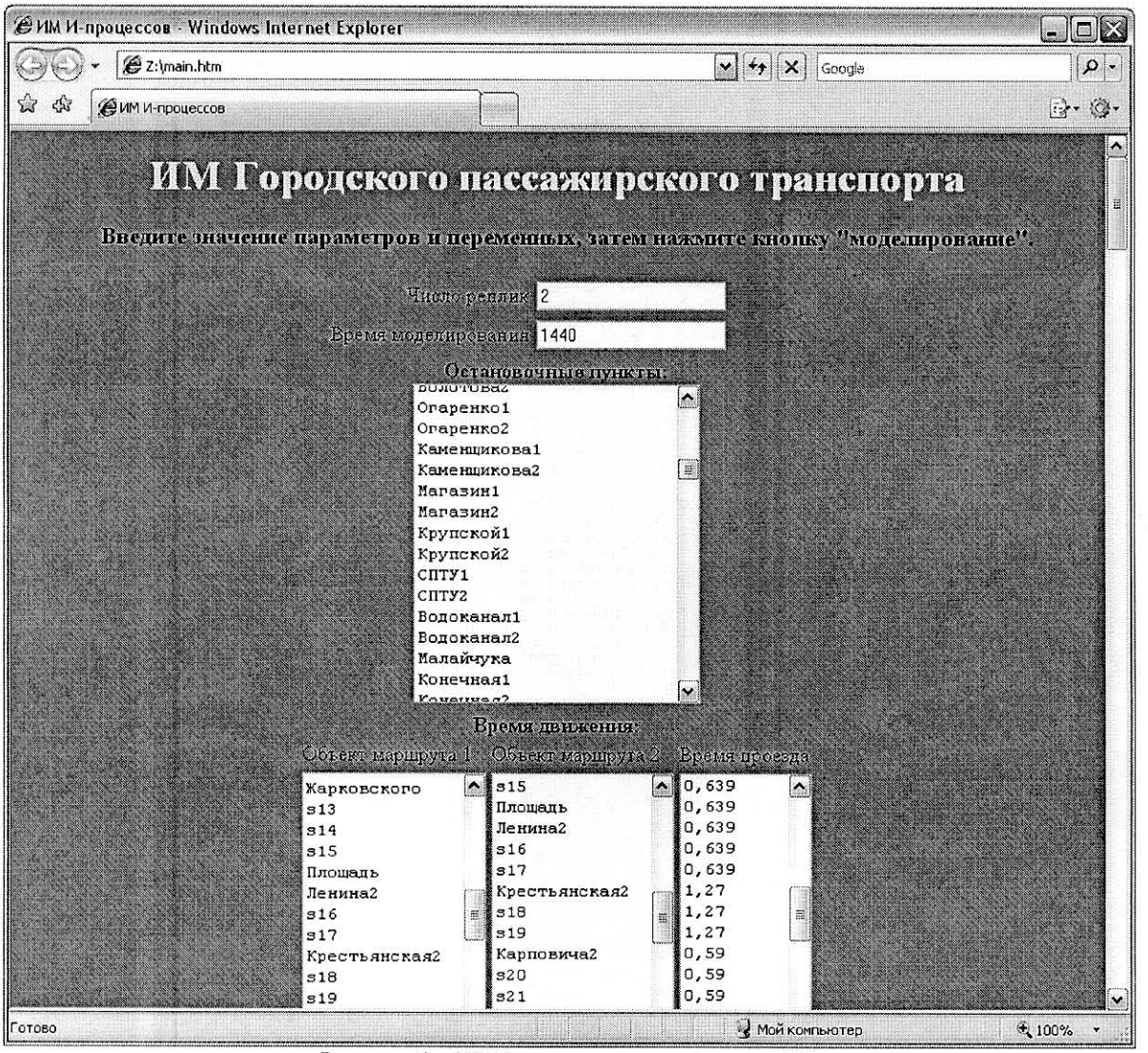


Рисунок 4 – WEB реализация программы ИМ

Третьим примером возможной реализации ИМ является использование WEB технологий. В этом случае программный модуль ИМ встраивается в Интернет сервер, а взаимодействие пользователя с ИМ происходит через сеть. При этом пользователь работает с ИМ через установленный у него браузер. Пример работы с ИМ через браузер Windows Internet Explorer 7 представлен на рисунке 4.

Многообразие доступных контейнеров позволяет обеспечить удобство работы заказчика с имитационной моделью в большинстве возможных случаев. На рисунке 5 в качестве примера отображена работа с программой ИМ технологических процессов сборочно-разборочного производства [1], для которой применен контейнер графического приложения Windows. На закладке «Входные данные» пользователь задает входные параметры и переменные имитационной модели, затем нажатием кнопки запускает моделирование. По окончании моделирования результаты помещаются в текстовое поле на закладке «Отклики».

Ещё одним из способов автоматизации работы с входными данными ИМ является использование вспомогательных редакторов входных данных со знакомым и дружественным пользователю интерфейсом. В качестве примера реализации такого способа автоматизации рассмотрим использование специально разработанного редактора параметров для имитационной модели городской маршрутной транспортной сети [2].

Входные данные программы ИМ городской маршрутной транспортной сети хранятся в текстовых файлах, для их редактирования может быть использован как произвольный текстовый редактор, так и редактор параметров, реализованный в среде табличного процессора Microsoft Excel. Подпрограммы на языке Visual Basic автоматически считывают и преобразуют входные данные при его запуске, а также предлагают их сохранение при закрытии редактора (рисунок 6).

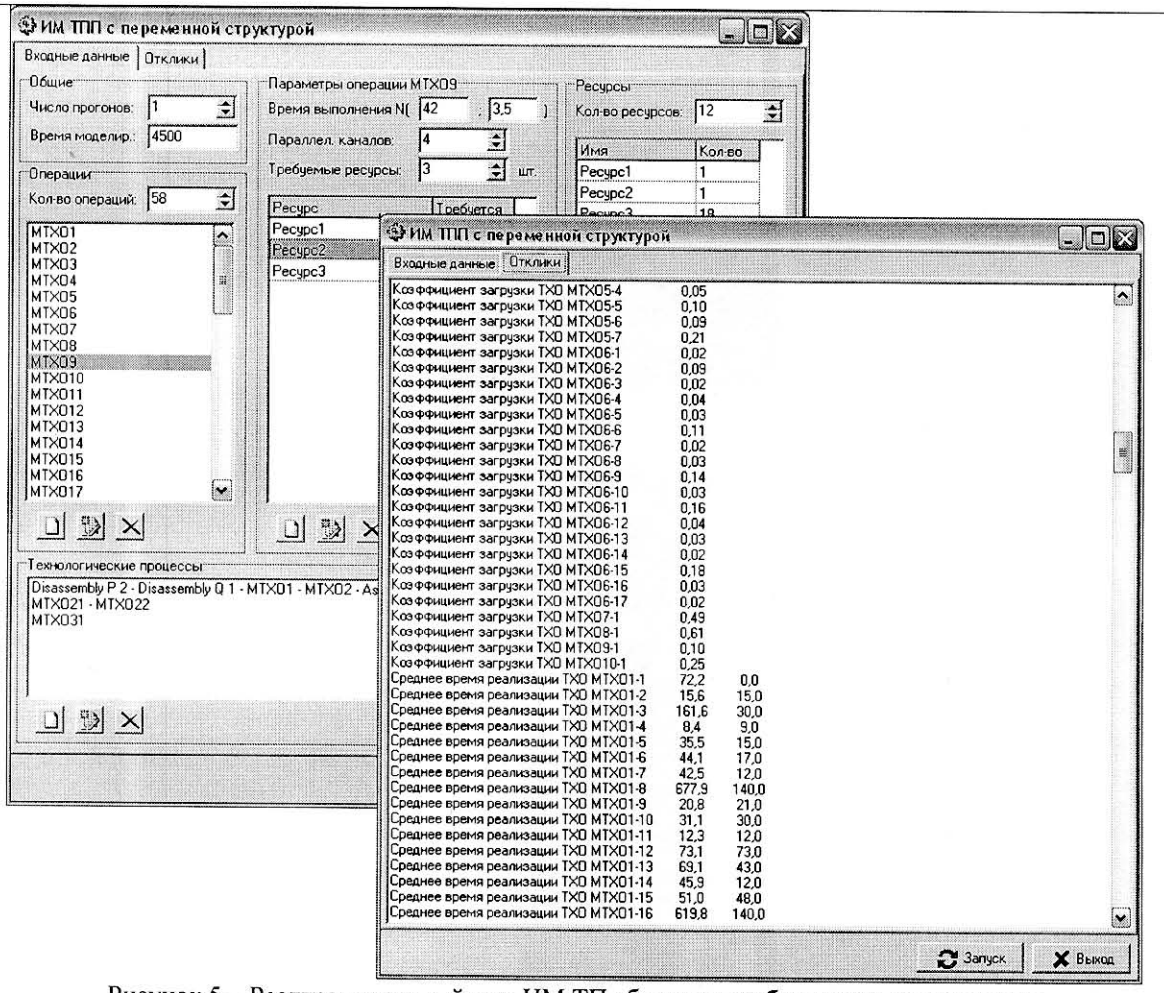


Рисунок 5 – Реализация контейнера ИМ ТП сборочно-разборочного производства

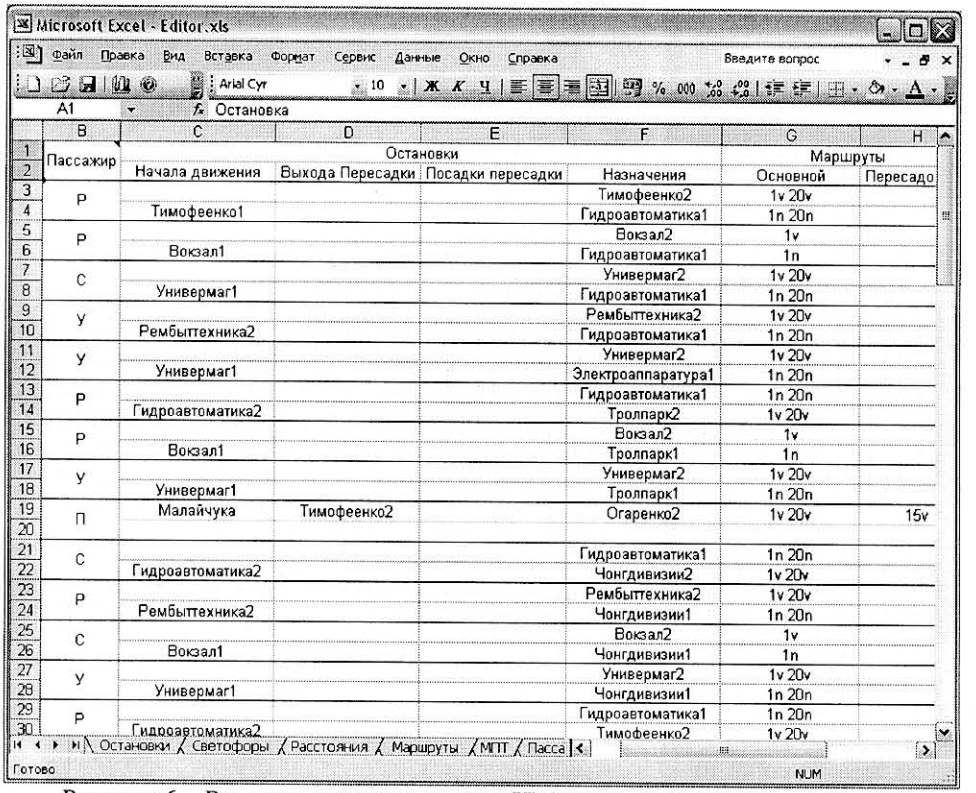


Рисунок 6 – Редактор входных данных ИМ городской транспортной сети

Результаты работы ИМ городской маршрутной транспортной сети помещаются в отдельные текстовые файлы, которые могут быть просмотрены в любом текстовом редакторе (рисунок 7) или открыты в табличном процессоре Microsoft Excel для дальнейшей обработки.

Также программа ИМ городской маршрутной транспортной сети может сохранять результаты в формате HTML, которые могут быть размещены на WEB-узле и быть просмотрены с помощью любого браузера, например Windows Internet Explorer (рисунок 8).

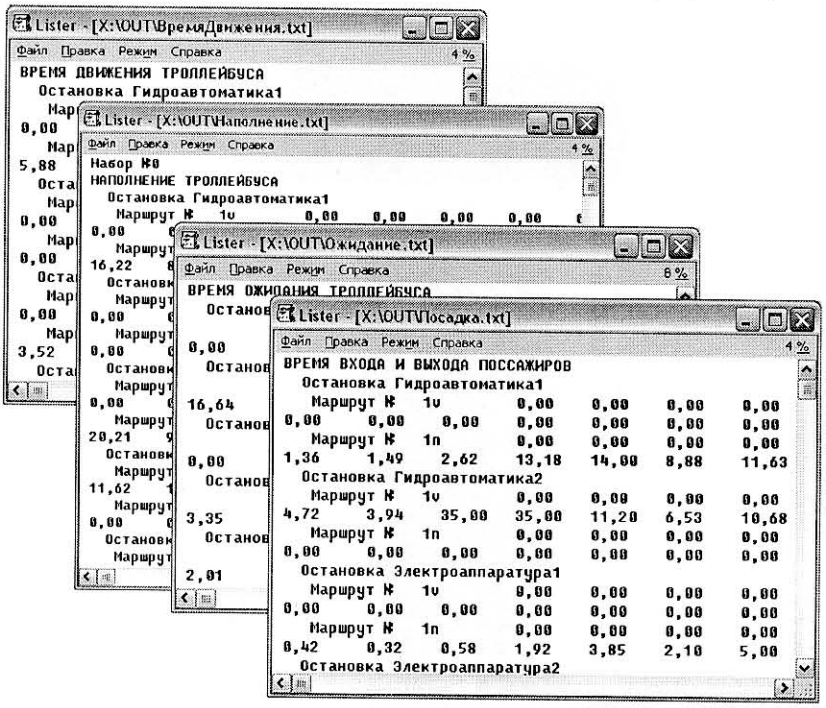
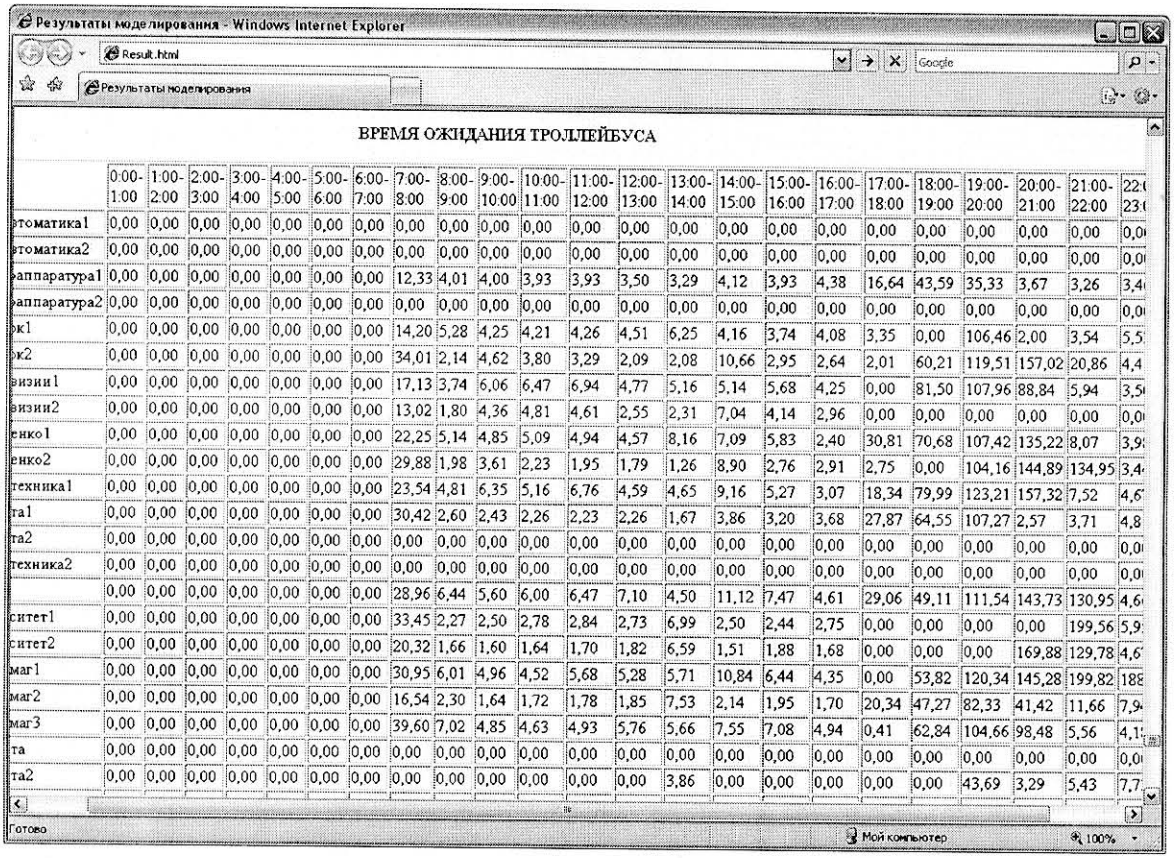


Рисунок 7 – Файлы откликов ИМ городской транспортной сети



	0:00-1:00	1:00-2:00	2:00-3:00	3:00-4:00	4:00-5:00	5:00-6:00	6:00-7:00	7:00-8:00	8:00-9:00	9:00-10:00	10:00-11:00	11:00-12:00	12:00-13:00	13:00-14:00	14:00-15:00	15:00-16:00	16:00-17:00	17:00-18:00	18:00-19:00	19:00-20:00	20:00-21:00	21:00-22:00	22(23(
втоматика1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
втоматика2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
аппаратура1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	12,33	4,01	4,00	3,93	3,93	3,50	3,29	4,12	3,93	4,38	16,64	43,59	35,33	3,67	3,26	3,4(
аппаратура2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
ка1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	14,20	5,28	4,25	4,21	4,26	4,51	6,25	4,16	3,74	4,08	3,35	0,00	106,46	2,00	3,54	5,5
ка2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	34,01	2,14	4,62	3,80	3,29	2,09	2,08	10,66	2,95	2,64	2,01	60,21	119,51	157,02	20,86	4,4
визия1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	17,13	3,74	6,06	6,47	6,94	4,77	5,16	5,14	5,68	4,25	0,00	81,50	107,96	88,84	5,94	3,5(
визия2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	13,02	1,80	4,36	4,81	4,61	2,55	2,31	7,04	4,14	2,96	0,00	0,00	0,00	0,00	0,00	0,0(
енко1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	22,25	5,14	4,85	5,09	4,94	4,57	8,16	7,09	5,83	2,40	30,81	70,68	107,42	135,22	8,07	3,9(
енко2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	29,88	1,98	3,61	2,23	1,95	1,79	1,26	8,90	2,76	2,91	2,75	0,00	104,16	144,89	134,95	3,4(
техника1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	23,54	4,81	6,35	5,16	6,76	4,59	4,65	9,16	5,27	3,07	18,34	79,99	123,21	157,32	7,52	4,6(
та1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	30,42	2,60	2,43	2,26	2,23	2,26	1,67	3,86	3,20	3,68	27,87	64,55	107,27	2,57	3,71	4,8
та2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
техника2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
	0,00	0,00	0,00	0,00	0,00	0,00	0,00	28,96	6,44	5,60	6,00	6,47	7,10	4,50	11,12	7,47	4,61	29,06	49,11	111,54	143,73	130,95	4,6(
ситет1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	33,45	2,27	2,50	2,78	2,84	2,73	6,99	2,50	2,44	2,75	0,00	0,00	0,00	0,00	199,56	5,9(
ситет2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	20,32	1,66	1,60	1,64	1,70	1,82	6,59	1,51	1,88	1,68	0,00	0,00	0,00	169,88	129,78	4,6(
маг1	0,00	0,00	0,00	0,00	0,00	0,00	0,00	30,95	6,01	4,96	4,52	5,68	5,28	5,71	10,84	6,44	4,35	0,00	53,82	120,34	145,28	199,82	188
маг2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	16,54	2,30	1,64	1,72	1,78	1,85	7,53	2,14	1,95	1,70	20,34	47,27	82,33	41,42	11,66	7,9(
маг3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	39,60	7,02	4,85	4,63	4,93	5,76	5,66	7,55	7,08	4,94	0,41	62,84	104,66	98,48	5,56	4,1(
та	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,0(
та2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	3,86	0,00	0,00	0,00	0,00	0,00	43,69	3,29	5,43	7,7(

Рисунок 8 – Отклик ИМ городской транспортной сети в формате HTML

Код программы модели, обеспечивающий экспорт откликов в формате HMTL находится в отдельном программном модуле, и может быть использован и с другими имитационными моделями, разработанными на основе системы моделирования MICIC 4 [3]. Также для моделей, реализованных на основе системы моделирования MICIC 4, разработаны специальные классы, обеспечивающие работу программы ИМ с файлами форматов XML, HTML и XLS взаимодействуют по следующей схеме (рисунок 9).

Рисунок 9 – Схема взаимодействия классов

2 Интерфейс взаимодействия контейнера и имитационной модели

Уровень структуры данных обслуживает интерфейс взаимосвязи MICIC4 с различными форматами входных и выходных данных. Этот уровень организуется посредством абстрагированных от MICIC4 классов для работы с документами форматов XML, XLS. Взаимосвязь между уровнем MICIC4 и уровнем структуры данных реализует класс *PlanOfExperiment*, который наследуется от базового класса системы MICIC4 *Experiment*. Базовыми классами для уровня структуры данных являются:

– *PlanOfExperiment* – «План Эксперимента», этот класс является наследником базового класса «Эксперимент» (*Experiment*) MICIC4;

– класс *PrmRetriever* используется для получения параметров, он включает методы, которые переопределены для XML и XLS формата в классах *XMLRetriever* и *XLSRetriever* соответственно;

– класс «Парсер» (*Parser*) совершает посимвольный разбор содержимого файла;

– класс«XLS Парсер» (*XLSParser*) создает дерево параметров;

– класс «Дерево» (*Tree*) представляет некоторую древовидную структуру (например, XML документ);

– класс «Генератор Дерева» (*TreeMaker*) создает дерево, определяющее структуру xml документа;

– класс «Узел» (*Nod*) представляет собой единицу (узел) древовидной структуры, реализованной классом *Tree*;

– класс «Атрибуты» (*Attributes*) определяет множество пар «ключ» – «значение».

Таким образом, работа с данными отделена от реализации самой модели. Причем работа с параметрами не зависит от типа файла, с которым работает пользователь. Для чтения и сохранения данных в Excel используется COM технология (Component Object Model).

Для обеспечения гибкости при разработке схемы взаимодействия классов были выделены три основных независимых слоя функционирования. Под независимостью понимается тот факт, что при использовании одного из слоев не обязательно знать, как устроены и функционируют нижние слои. Самый верхний слой имеет узкий интерфейс, который с одной стороны легок для использования и не требует много времени для изучения, а с другой стороны обеспечивает достаточный набор средств для реального применения. Диаграмма базовых классов представлена на рисунке 10.

Классы верхнего слоя предназначены для использования в СМ MICIC4. Классы среднего уровня предоставляют интерфейс для работы с данными из файлов в формате XML и XLS и формирования документов в указанных форматах. Нижний уровень представлен следующими классами:

– классы, позволяющие создавать необходимые структуры данных (ассоциативные массивы для хранения атрибутов тегов, деревья для представления XML документов);

– классы, обеспечивающие обмен данными с XLS файлами;

– классы, производящие разбор XML и XLS документов;

– классы для формирования отчетов (таблиц) в формате HTML и XLS.

Функциональное назначение уровней приведено на рисунке 11.

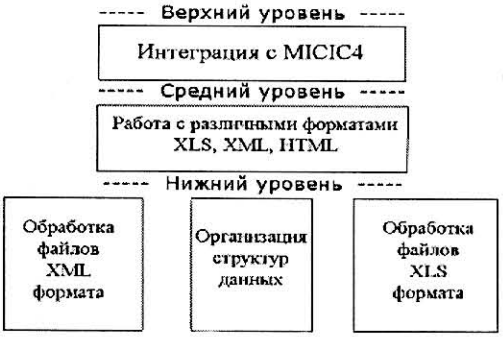


Рисунок 11 – Функциональное назначение уровней

Верхний уровень представлен всего одним классом *PlanOfExperiment*. Предлагается следующая модель использования данного класса:

– описывается наследник от класса *PlanOfExperiment*;

– в наследнике переопределяется виртуальный метод *execute*, именно в данном методе будут производиться все необходимые действия, и формироваться отчет;

– при работе с экземпляром нового класса необходимо проинициализировать обработчик отчета.

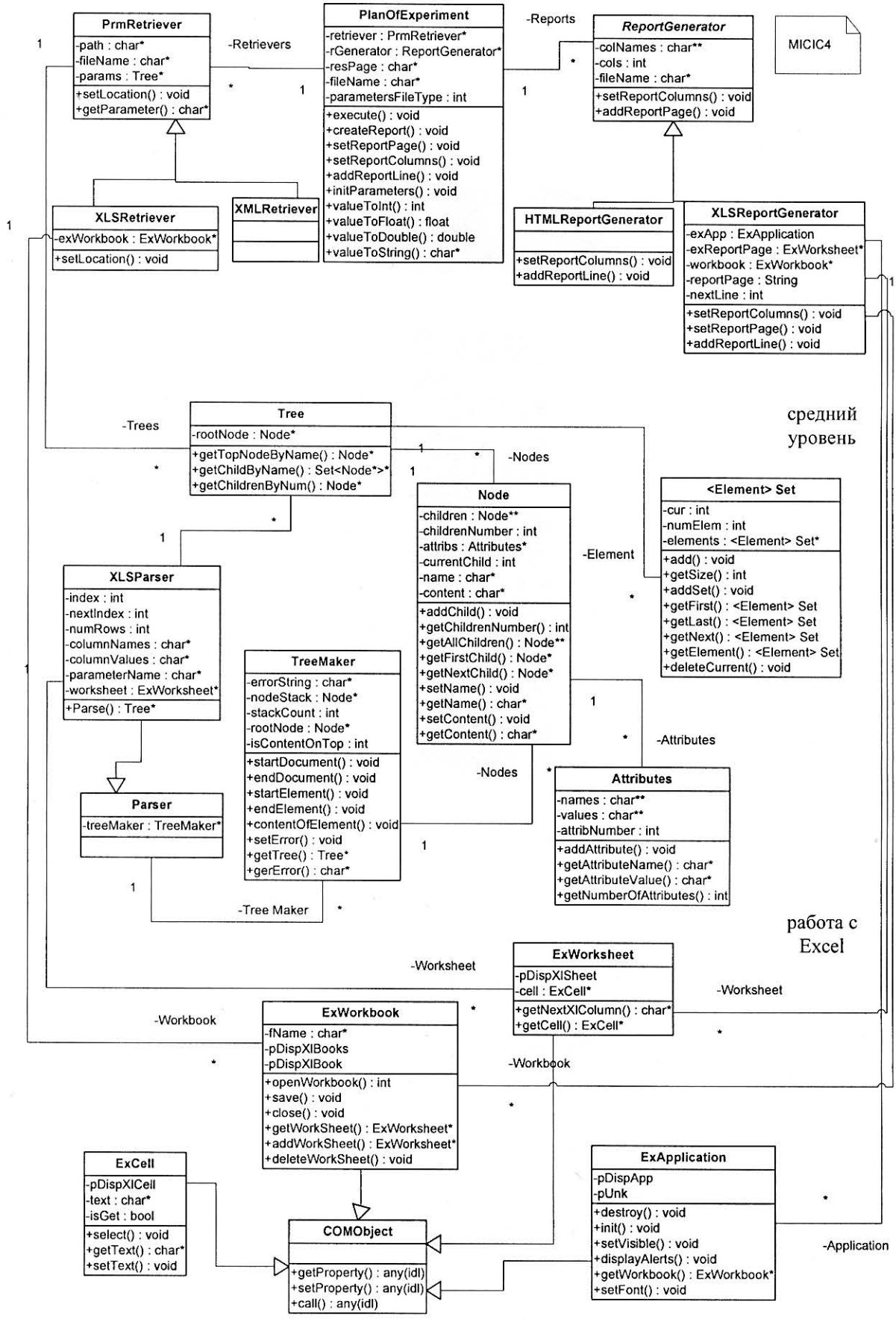


Рисунок 10 – Диаграмма базовых классов

Пример использования класса *PlanOfExperiment* приведен ниже.

```
class TestPlan : public PlanOfExperiment
{ public:
  TestPlan (char* name, int type):
    PlanOfExperiment (name, type){}
  void execute() {
    char line[3][256];
    int nextRandom;
    for(int i = 0; i < valueToInt("size");
        i++) {
      nextRandom = rand();
      sprintf(line[0], "%d", i);
      sprintf(line[1], "%d", nextRandom);
      sprintf(line[2], "%s",
      nextRandom&1 ? "True":"False");
      addReportLine(line);
    }}
};
```

Рассмотрим интерфейс, предлагаемый классом *PlanOfExperiment*.

```
initParameters(char * fileName, int
type = USEXLS, char* location =
"Parameters");
```

Данный метод класса получает два параметра: имя файла, в котором находятся параметры; тип файла.

Второй параметр может принимать значения: USEXML, USEXLS.

Данные константы определены в этом же классе. Третий параметр указывает, откуда нужно считывать параметры. Для рабочей книги табличного процессора Microsoft Excel – это имя страницы, а для XML – последовательность тегов, которая ведет к тегу, в который вложены параметры. В методе происходит инициализация и считывание параметров из указанного файла. После вызова пользователь получает возможность работать с извлеченными параметрами. Для этого существует серия методов:

```
- int valueToInt (char* name);
- float valueToFloat (char* name);
- double valueToDouble (char* name);
- const char* valueToString (char*
name);
```

Все эти функции получают в качестве параметра строку, которая является именем извлеченного из файла параметра. Для проведения всех необходимых действий предлагается функция *execute*. Данная функция является виртуальной. Также в классе *PlanOfExperiment* реализована возможность создавать таблицу отчета. Для этого необходимо вызвать функцию *void createReport (char* fName, int type=USEXLS)*, которой передается имя файла, который нужно использовать и его тип. В данном случае доступными являются два типа: USEXLS, USEHTML. Они позволяют сохранять данные в форматах XLS и HTML соответственно. Таблица результатов должна иметь столбцы. Их имена и количество задается при помощи функции *void setReportColumns (char[][256], int count)*. Для занесения информации в таблицу существует функция *void addReportLine (char[][256])*, которая добавляет новую строку в таблицу. Аргумент функции, как и в случае с предыдущей функцией, является массивом строк. Каждый элемент массива записывается в соответственную колонку таблицы. Также в интерфейсе класса *PlanOfExperiment* существует функция, предназначенная для работы только лишь с форматом XLS: *void setReportPage (char* page)*. При работе с книгой табличного процессора Microsoft Excel данная функция устанавливает лист, на котором будет формироваться таблица отчета. Если такой лист не существует, то он будет создан, иначе он будет очищен.

Средний уровень представлен более разнообразным числом классов по сравнению с верхним уровнем. Сюда входят семь классов для высокоуровневой работы с получением данных из файлов разных форматов и сохранения полученных результатов.

Класс *XLSParser* предназначен для разбора структуры страницы Excel. Именно этот класс работает с содержимым страницы. Это означает, что если потребуется изменение формата файла, то нужно будет всего лишь заменить этот класс, оставив при этом его интерфейс. Конструктор данного класса *XLSParser (ExWorksheet* sheet)* получает указатель на объект ExWorksheet, из которого можно получить содержимое страницы. Основной метод класса *Tree* Parse ()*, непосредственно в нем и происходит разбор содержимого страницы. Сама страница должна удовлетворять условиям, указанным в описании шаблона. Метод возвращает указатель на дерево.

Метод *static char* getNextXlColumn (char* col)* используется для получения следующего имени столбца по предыдущему согласно именования, принятого в табличном процессоре Microsoft Excel (столбцы именуются от ‘A’ до ‘Z’, а затем ‘AA’, ‘AB’, ‘AC’, …).

Класс *PrmRetriever* предоставляет интерфейс для всех классов, которые считывают параметры из файлов разного типа. Конструктор *PrmRetriever (char* fName, char* location)* получает в качестве первого аргумента имя файла, с которым будет осуществляться вся работа. Второй аргумент – это местоположение параметров в файле. Данная строка указывает путь таким же образом, как указывается путь к файлу с помощью имен каталогов, в которые он вложен. Например «*project/parameters/section[1]/cell[1][2]*». Для разделения используется символ «/».

Основная функция, используемая в классе *PrmRetriever*, это *char* getParameter (char* name)*. Она получает в качестве аргумента имя параметра и возвращает его значение.

Класс *PrmRetriever* имеет два наследника для работы с разными форматами файлов. Класс XLSRetriever предназначен для работы с файлами табличного процессора Microsoft Excel. В

интерфейс родительского класса он ничего нового не вносит. Все особенности работы с форматом XLS скрыты в конструкторе данного класса.

Класс *XMLRetriever* организует работу с файлами в XML формате. Он также не расширяет интерфейс родительского класса. Следует отметить, что для данного класса можно вызывать *setLocation* во время работы приложения, что приведет к корректной смене положения параметров. Такая возможность объясняется тем, что в данном классе хранится дерево всего XML документа и смена ветки, из которой делается выбор параметра, пройдет безошибочно.

Класс *ReportGenerator* предназначен для формирования таблицы. Он имеет простой и удобный интерфейс для составления таблиц отчетов. Это виртуальный класс, который не привязан к определенному формату файлов. Рассмотрим методы класса. Конструктор *ReportGenerator (char* fileName)* имеет всего один аргумент – имя файла, в котором будет находиться таблица. Функция *virtual void setReportColumns (char cols[][256], int count)* предназначена для задания имен столбцов и их количества. Имена столбцов хранятся во внутренней переменной *colNames*, а количество столбцов – в *cols*. Эти переменные необходимы лишь при создании наследников данного класса, которые будут организовывать сохранение таблицы в файле определенного формата. Метод *virtual void addReportLine (char line[][256]) = 0* вызывается для добавления очередной строки в таблицу. Он является чисто виртуальным и должен быть реализован для каждого конкретного формата. Интерфейс данного класса является в большинстве случаев достаточным для построения таблиц.

Класс *HTMLReportGenerator* является наследником класса *ReportGenerator* и служит для формирования таблицы в виде HTML страницы. В результате его работы формируется таблица в HTML файле, который открывается любым браузером (рисунок 8).

Класс *XLSReportGenerator* является наследником класса *ReportGenerator* и служит для создания таблицы в формате Excel. Данный класс добавляет всего один метод к интерфейсу родительского класса *ReportGenerator void setReportPage (char* page)*.

Из рисунка 11 видно, что ***нижний уровень*** состоит из трех наборов классов:

– классы для представления структур данных;

– классы для работы с XML документами;

– классы для работы с программой Excel.

Классы для представления структур данных. Класс *Attributes* хранит в себе множество пар вида «ключ = значение». Он имеет два конструктора:

– конструктор для создания пустого экземпляра класса;

– конструктор с начальной инициализацией множеством имен атрибутов и значений.

Реализация этого класса необходима для полноценного представления узла дерева XML файла. Также в интерфейсе имеются методы для получения значения имени атрибута и его значения по номеру.

```
char* getAttributeName (int attribNum);
char* getAttributeValue(int attribNum);
```

Количество атрибутов можно получить при вызове метода

```
int getNumberOfAttributes();
```

Класс *Node* представляет собой элемент структуры XML файла. Легко представить XML структуру в виде дерева. Основой дерева будет внешний тег. Теги, содержащиеся внутри, представляют собой его ветви и в тоже время основу для других ветвей. Каждый узел данного дерева представляется объектом типа *Node*. Каждый такой объект может содержать параметры. Класс *Node* имеет два конструктора:

```
Node();
Node(char* nm, Attributes* attr, char*
cont);
```

Первый аргумент – это имя тега, второй – атрибуты тега, третий – содержимое. Также существуют *get* и *set* методы для всех трех составляющих узла дерева. В классе определен ряд методов для работы с дочерними тегами. Метод *void addChild (Node* child)* добавляет объект *child* к узлу. Метод *int getChildrenNumber ()* возвращает число дочерних элементов. Метод *const Node** getAllChildren ()* возвращает указатель на массив всех дочерних элементов. Метод *Node* getFirstChild ()* возвращает указатель на первый элемент в списке всех дочерних элементов. Метод *Node* getNextChild ()* возвращает очередной дочерний элемент в списке. По достижению конца – возвращает NULL. При вызове метода *getFirstChild* обход списка начинается с начала.

Класс *Set* является множеством любых элементов. Он является шаблоном *template <class Element> class Set*. Этот класс имеет методы для добавления одного нового элемента в конец списка *void add(Element)*. К тому же реализована возможность добавлять целый список в конец существующего с помощью метода *void addSet (Set<Element>* set)*. Также данный класс имеет методы для получения первого и последнего элемента и элемента по номеру. Существует возможность перебора всех элементов по очереди с помощью метода *Element getNext (const int direction = FORWARDS)*. В качестве аргумента данный метод получает направление движения при просмотре списка, по умолчанию – вперед (от первого к последнему элементу). Также можно удалять текущий элемент с помощью метода *void deleteCurrent ()*.

Класс *Tree* используется не только для представления структуры XML документа, но и для хранения параметров. Конструктор класса

Tree (Node rootNd)* получает в качестве аргумента указатель на корневой элемент дерева. Класс имеет два статических метода:

– *static Set<Node*> *getChildByName (char *name, Node* node)*;

– *static Node* getChildrenByNum (int num, Node* node)*;

Первый возвращает множество узлов дерева, являющихся дочерними узла *node* и имеющих имя *name*. Этот метод активно используется для поиска элементов во всем дереве. Так как на одном уровне и в одном узле могут находиться несколько дочерних элементов с одним именем, то данный метод возвращает именно множество, а не один элемент. Причем гарантируется, что в самом множестве все элементы будут содержаться в том же порядке, в котором они встречаются в дочерних элементах указанного узла *node*. Второй метод возвращает элемент по номеру среди дочерних элементов узла *node*.

Метод *Node* getTopNodeByName (char* namePath)* возвращает первый элемент в списке элементов, удовлетворяющих пути *namePath*.

Следующие два класса отвечают за организацию работы с файлами XML формата. Класс *TreeMaker* предназначен для формирования дерева. Он используется совместно с классом *Parser*, который занимается разбором и выделением синтаксически конструкций в файле формата XML. Конструктор класса Parser получает в качестве аргументов имя файла и указатель на объект типа *TreeMaker*. Класс *Parser* имеет всего одни метод *Parser (Document* doc, TreeMaker* tmaker*, в котором происходит весь разбор. В начале разбора документа вызывается метод *void startDocument (char* information)*, а при окончании разбора *void endDocument ()*. Когда в процессе разбора встречается открывающийся тег, вызывается метод объекта *TreeMaker void startElement (char* name, Attributes* attribs)*. Сам этот объект ведет стек открытых тегов, и при вызове данного метода добавляет имя открывшегося тега в стек. Одновременно с этим он ведет стек элементов *Node*. В этом методе создается объект типа *Node*, заносятся в него имя и атрибуты, и этот новый элемент добавляется в дочерние элементы для текущего *Node*. Новый *Node* при этом становится текущим. Когда в результате разбора выделяется строка, являющаяся содержимым тега, вызывается метод *void contentOfElement (char* content)*. В этом методе в текущем элементе в стеке устанавливается содержимое, затем, при обнаружении закрывающегося тега, вызывается метод *void endElement (char* name)*. При вызове данного метода происходит проверка совпадения верхнего имени открытого тега с именем закрывающегося. Если они не совпадают, то устанавливается ошибка, текст которой может быть получен вызовом метода *char* getError ()*. Далее в методе *endElement* происходит извлечение верхних элементов из стеков. Таким образом, текущим элементом становится родительский тег по отношению к извлеченному. После этого с помощью метода класса *TreeMaker Tree* getTree ()* можно получить сформированное дерево.

Для работы с *табличным процессором Microsoft Excel* посредством технологии COM был описан класс *ExcelCOMWorker*. Этот класс использует класс *COMObject*, который позволяет взаимодействовать с табличным процессором посредством методов, реализующих следующую функциональность:

– вызов функции указанного объекта по ее имени;

– получение свойства объекта по его имени;

– установка свойства объекта по его имени.

Диаграмма классов при работе с табличным процессором Microsoft Excel представлена на рисунке 12.

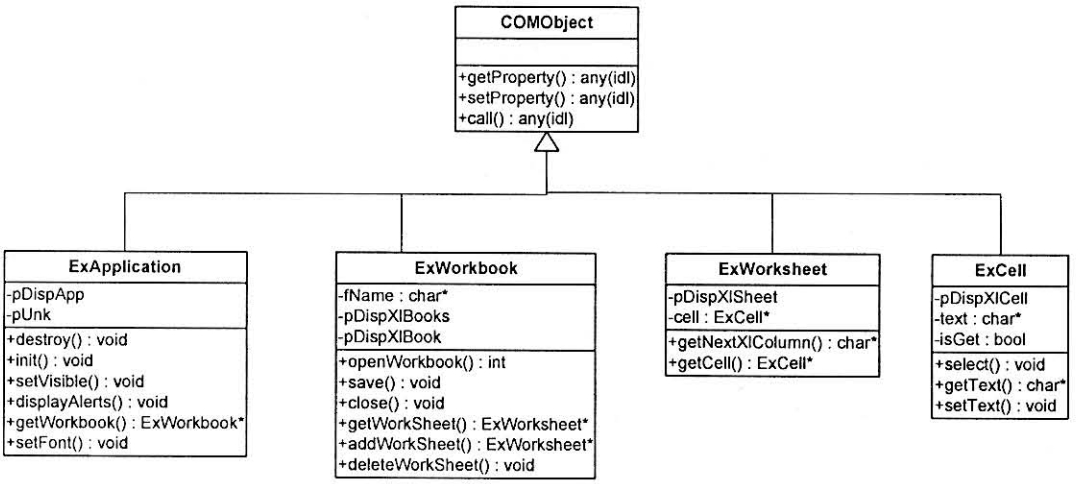


Рисунок 12 – Диаграмма классов по работе с документом Excel

Пример размещения информации на листе в рабочей книге табличного процессора приведен на рисунке 13.

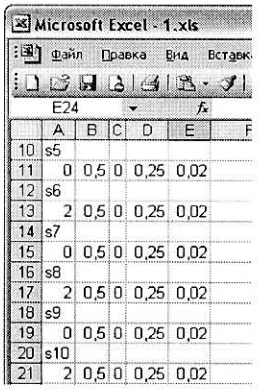


Рисунок 13 – Размещение входных данных в табличном процессоре Microsoft Excel

Использование технологии COM позволяет как считывать данные из ячеек листа, так и заносить результаты работы ИМ на листы табличного процессора (рисунок 14).

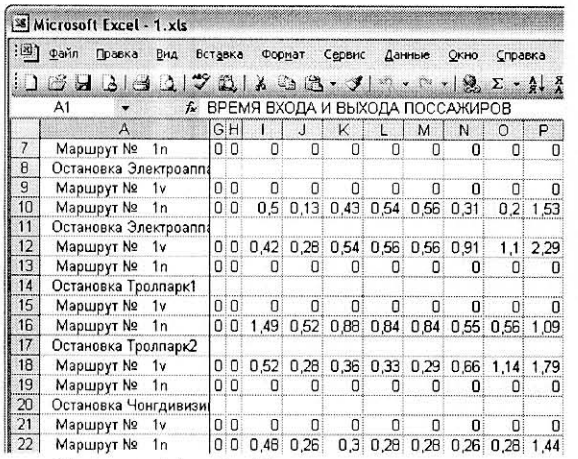


Рисунок 14 – Отображение результатов моделирования в табличном процессоре Microsoft Excel

Заключение

В статье рассмотрены вопросы интеграции программ имитационных моделей в информационную среду заказчика. На примере разработанных в системе моделирования MICIC 4 имитационных моделей показаны возможные способы реализации контейнеров ввода/вывода и модулей преобразования формата результатов. Использование контейнеров и преобразование форматов вывода позволяет упростить работу пользователя с моделью, сделать её более «дружественной», а также ускорить подготовку имитационных экспериментов и предварительную обработку результатов моделирования. Преимуществом контейнеров является и то, что при использовании контейнеров пользователь максимально использует для работы привычное для него программное обеспечение.

ЛИТЕРАТУРА

1. *Чечет, П.Л.* Декомпозиция технологических процессов производства с иерархической структурой / П.Л. Чечет // Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях : материалы VII Республиканской научной конференции студентов и аспирантов 22–24 марта 2004 г. / ГГУ им. Ф. Скорины. – Гомель, 2004. – С. 73–74.

2. *Чечет, П.Л.* Реализация имитационной модели сети городского пассажирского транспорта / П.Л. Чечет // Известия ГГУ им. Ф. Скорины. – Гомель. – 2006. – №4 (37). – С. 102–104.

3. *Левчук, В.Д.* Программно-технологические комплексы имитации сложных дискретных систем / В.Д. Левчук, И.В. Максимей. – Гомель : ГГУ им. Ф. Скорины, 2006. – 263 с.

Поступила в редакцию 10.03.10.