УДК 004.942
DOI: 10.15827/0236-235X.129.013-019
Дата подачи статьи: 18.07.19
2020. Т. 33. № 1. С. 013–019

Разработка специального программного обеспечения для ввода исходных данных при имитационном моделировании

С.А. Чернышев [1,2], *к.т.н., старший научный сотрудник, ст. преподаватель, chernyshev.s.a@bk.ru*
С.А. Антипова [1], *к.ф.-м.н., старший научный сотрудник, samiraspb11@gmail.com*

[1] *Военная академия материально-технического обеспечения им. генерала армии А.В. Хрулева, г. Санкт-Петербург, 199034, Россия*
[2] *Санкт-Петербургский государственный университет аэрокосмического приборостроения, г. Санкт-Петербург, 190000, Россия*

Широкое внедрение цифровых технологий во все сферы человеческой деятельности порождает возрастающий поток структурированной и неструктурированной информации, поступающей из большого количества различных, в том числе разрозненных или слабосвязанных, источников информации. Особенно остро стоит вопрос об упорядочении, целенаправленном структурировании информации и данных для последующего их использования при проведении имитационного моделирования различных сложных технических процессов.

В статье анализируются подходы, позволяющие упростить процессы ввода и редактирования значительного массива исходных данных, необходимых для разработки сложных имитационных моделей. Так, непосредственная работа с файлами текстового и табличного форматов предполагает частые изменения структуры параметров модели с минимальными либо отсутствующими аналитическими проверками вводимых параметров. Работа с файлом, хранящим исходные параметры, через разработанную структуру промежуточного представления данных позволяет выполнять необходимые обращения к хранящимся в файле данным, но является трудоемкой и нецелесообразной.

Наиболее удачным и универсальным вариантом, по мнению авторов, является работа с файлом, хранящим исходные параметры, через промежуточную БД. При таком подходе БД может как выступать промежуточным звеном в заполнении и редактировании файлов с исходными параметрами модели, так и являться источником исходных данных для проведения моделирования. Приоритетными для использования в данном случае являются нереляционные БД NoSQL, обеспечивающие горизонтальную масштабируемость, гибкость и высокую производительность.

Ключевые слова: *имитационное моделирование, объектно-ориентированное программирование, база данных, формат данных.*

С развитием современных информационных технологий разработка любого изделия, технологического процесса и т.д. начинается с его моделирования. Это позволяет значительно уменьшить затраты при изготовлении конечного продукта, проработать возможности его качественного улучшения и смоделировать процессы, протекающие как в самом продукте (системе), так и в операционном окружении, где он будет использоваться. Такой подход позволяет лучше понять протекающие процессы, их взаимодействие и имеющиеся между ними зависимости, которые, на первый взгляд, не так очевидны.

В числе важных задач при таком подходе к разработке целевого продукта – подбор, структурирование, ввод, хранение и редактирование исходных параметров для проводимого моделирования. Если у модели мало исходных параметров, их можно хранить в .txt-, .csv- или excel-файлах, а также в виде локальных переменных в самой модели. Для их изменения и дополнения не требуется значительных усилий.

При моделировании сложного в техническом плане изделия или при проведении имитационного моделирования процессов различной сложности [1–3] количество исходных параметров значительно возрастает и может достигать нескольких сотен и даже тысяч. Например, изменения в современном характере военных действий обусловлены, в частности, масштабностью системы, включающей свыше десятка основных процессов системного уровня, которые при дальнейшей детализации моделирования декомпозируются в сотни процессов, а декомпозиция процессов усложнена сильной взаимосвязанностью по общим внутренним ресурсам, включая наряд сил

и средств противоборствующих сторон. При имитационном моделировании требуется всесторонний учет установленных нормативных требований и порядка применения сил и средств, что накладывает особые требования к порядку формирования и ввода исходных данных. С данной проблемой авторы столкнулись при разработке имитационной модели процесса ракетно-технического обеспечения в программной среде AnyLogic при выборе максимального уровня детализации моделируемого процесса.

В таких случаях подходы к вводу и редактированию исходных параметров моделируемой системы должны коренным образом отличаться от традиционного хранения в текстовых и табличных файлах. Например, изменить исходные параметры модели или расширить их наименования может только тот, кто знаком со структурой документа, предельными значениями изменяемых параметров и т.д. В силу этого оптимальным решением возникающей проблемы будет разработка ПО, которое позволит выполнять ввод и редактирование исходных параметров для проведения моделирования. Помимо этого, на разрабатываемое *специальное ПО* (СПО) будут ложиться задачи проверки корректности ввода данных. Основная задача инженера-программиста перед проведением экспериментов в рамках имитационного моделирования – нахождение в рамках предельных значений и упрощение ввода текущих значений на основе уже имеющихся данных.

Такой подход упрощает формирование нового файла либо БД с исходными параметрами или редактирование уже имеющегося, что позволяет привлекать к решению этой задачи не только оператора, разбирающегося в структуре данных исходных параметров моделируемой системы, но и человека, не обладающего специальными навыками в программной инженерии.

Целью данной статьи является описание возможных подходов к разработке СПО для задач ввода и редактирования исходных параметров, используемых при имитационном моделировании, их положительных и отрицательных сторон, а также области применения.

Подходы к разработке СПО ввода и редактирования исходных параметров

Предлагаемые подходы к разработке ПО данной направленности можно распределить следующим образом:

– непосредственная работа с файлом форматов .txt, .csv, .xlsx, хранящим исходные параметры, используемые при моделировании;

– работа с файлом, хранящим исходные параметры, через промежуточную БД;

– работа с файлом, хранящим исходные параметры, через разработанную структуру промежуточного представления данных, которая позволяет выполнять необходимые обращения к хранящимся в файле данным наподобие SQL-запросов.

Последний подход является самым сложным и долгим в реализации. Его можно отнести к разделу академических проектов, цель которых – наработка бакалаврами или магистрами компетенций в области разработки ПО. В связи с этим в данной работе он не будет рассматриваться. Далее представлен анализ первых двух подходов с примерами исходных кодов, написанных на языке программирования C++ [4], с использованием кроссплатформенного фреймворка Qt [5].

Непосредственная работа с файлом, хранящим исходные параметры

Данный случай рассмотрен на примере работы с файлом .xlsx-формата (Excel). Excel-файл представляет собой книгу с листами, каждый из которых имеет собственное название и представляет собой таблицу $N \times M$, где N – количество строк, а M – количество столбцов. Каждому столбцу листа назначается переменная, за которую он будет отвечать. Чтобы можно было разобраться в структуре листа, принято в первой строке каждому столбцу, по количеству переменных, задавать названия (рис. 1). Последующие строки используются для ввода значений.

№ п/п	Наименование	Дислокация	Широта	Долгота	АБ	ДТ	АК	Налив
1								
2								

Рис. 1. Пример структуры листа Excel-файла с назначенными столбцам переменными

Fig. 1. The structure example of Excel-file with the appointed variable columns

Для создания, редактирования и форматирования Excel-файлов при работе в связке C++ и Qt чаще всего используется кроссплатформенная библиотека qtxlsx [6]. Единственным минусом ее использования является отсутствие

возможности удалять строки без написания дополнительного кода в исходные коды самой библиотеки. Это связано с тем, что изначально библиотека направлена на последовательное наполнение создаваемого файла ПО, в котором она используется. Таким образом, функционал по редактированию файлов несколько ограничен.

Приведем пример структуры листа Excel-файла (рис. 1), описанной на языке программирования C++ с использованием механизма перечислений и словаря:

```
enum class CombinesEnum {Name=1,Position,Latitude, Longitude, AB, DT, AK, Filling, _COUNT};

inline QString enumToString(CombinesEnum v)
{
   const QMap<CombinesEnum,QString> combinesEnumString{
        {CombinesEnum::Name, "Наименование"},
        {CombinesEnum::Position, "Дислокация"},
        {CombinesEnum::Latitude, "Широта"},
        {CombinesEnum::Longitude, "Долгота"},
        {CombinesEnum::AB, "АБ"},
        {CombinesEnum::DT, "ДТ"},
        {CombinesEnum::AK, "АК"},
        {CombinesEnum::Filling, "Налив"}
      };
      QMap<CombinesEnum,QString>::const_iterator it = combinesEnumString.find(v);
      return it == combinesEnumString.end() ? "Out of range" : it.value();
}
```

При таком подходе к описанию колонок электронной таблицы ее создание и заполнение первой строки с перечислением названий параметров каждой из колонок можно осуществлять следующим образом:

```
m_tempDoc->addSheet(enumToString(SheetEnum::Combines));
for(int i = 1; i< static_cast<int>(CombinesEnum::_COUNT); i++ ){
      m_tempDoc->write(1,i,enumToString(static_cast<CombinesEnum>(i)));
}
```

Отсюда следует, что для простого создания нового Excel-файла необходимо представить описание всей структуры документа (количество и названия листов) и параметров каждой электронной таблицы (листа) в виде перечислений.

Стоит обратить внимание на интерфейс для работы с каждым из листов создаваемого или редактируемого документа. Ввиду того, что у каждого листа собственное количество параметров, то есть различное количество столбцов, с которыми осуществляется работа, интерфейс должен предоставлять максимальную абстракцию для работы с листами. Этого можно добиться, используя при реализации интерфейсного класса для метода записи и чтения текущей строки тип QJsonObject с данными в формате JSON [7]. Формат JSON наиболее удобен при взаимодействии с JavaScript, вместе с тем почти все языки программирования имеют специализированные библиотеки для преобразования объектов в данный формат. Приведем пример данных в формате JSON:

```
{
 "Array": [
    true,
    999,
    "string"
 ],
 "key": "value",
 "null": null
}
```

Наследование от интерфейсного класса происходит следующим образом:

```
 class SheetInterface: public QObject{
Q_OBJECT

public:
  explicit SheetInterface(QObject *parent = nullptr): QObject(parent){}
  virtual ~SheetInterface(){}

  virtual void setWorkSheet(QXlsx::Worksheet* sheet)=0;
  virtual int getRowsAmount() = 0;
  virtual const QJsonObject& getValues(int row)=0;
  virtual void writeValues(const QJsonObject& data, int row)=0;
  virtual void deleteRow(int i)=0;

public slots:
  virtual void openFile(){}
  virtual void newFile(){}

signals:
  void rowsAmounUpdate(int maxRows);
};
```

Затем необходимо переопределить виртуальные методы. Чтение из текущего листа может осуществляться следующим образом:

```
const QJsonObject &CombinesSheet::getValues(int row)
{
   m_data[enumToString(CombinesEnum::Name)]= m_combinasSheet>read(row,static_cast<int>(CombinesEnum::Name)).toString();
   m_data[enumToString(CombinesEnum::Position)]= m_combinasSheet>read(row,static_cast<int>(CombinesEnum::Position)).toString(); m_data[enumToString(CombinesEnum::Latitude)]=m_combinasSheet>read(row, static_cast<int>(CombinesEnum::Latitude)).toDouble();
   m_data[enumToString(CombinesEnum::Longitude)]= m_combinasSheet>read(row,static_cast<int>(CombinesEnum::Longitude)).toDouble();
   m_data[enumToString(CombinesEnum::AB)]= m_combinasSheet>read(row,static_cast<int>(CombinesEnum::AB)).toDouble(); m_data[enumToString(Combi-nes
```

```
Enum::DT)]=m_combinasSheet>read(row,static_cast
<int>(CombinesEnum::DT)).toDouble();
    m_data[enumToString(CombinesEnum::AK)]=
m_combinasSheet>read(row,static_cast<int>(CombinesE-
num::AK)).toDouble();
    m_data[enumToString(CombinesEnum::Filling)]=
m_combinasSheet>read(row,static_cast<int>(CombinesE-
num::Filling)).toDouble();
      return  m_data;
    }
```

Очевидно, что такой подход позволяет считывать и записывать данные в произвольном порядке, но обладает существенным недостатком – при значительном масштабировании параметров или при изначально большом количестве параметров в текущем листе код разрастается и становится нечитаемым, значительно возрастает сложность его поддержки. В случае, если все значения столбцов в строке одного типа или их значительная часть расположены последовательно, для их чтения или записи можно использовать циклы:

```
    for (int i = static_cast<int>(SICHEnum::Connection);
i <= static_cast<int>(SICHEnum::_COUNT); ++i) {
          m_data[enumToString(static_cast
<SICHEnum>(i))] = m_sichSheet->read(row,i).toDou-
ble();
    }
```

В случаях, когда параметры перемешаны по типам данных и нет возможности их структурировать таким образом, чтобы работа с полями осуществлялась способом, представленным ранее, без потери смысловой составляющей, необходимо применять другой подход. Он заключается в использовании цикла с вложенной в него конструкцией switch-case. Это позволяет сгруппировать поля по типам данных и упростить их чтение и запись. Чтобы в одном методе сразу осуществлялись две операции (чтение и запись), необходимо в класс добавить еще один метод:

```
    private:
      void readWrite(int row,const QJsonObject& data =
QJsonObject());
      QXlsx::Worksheet* m_rpgSheet{nullptr};
      QJsonObject m_data;
```

В случае чтения данных вызов метода readWrite должен осуществляться следующим образом:

```
    const QJsonObject &RPGSheet::getValues(int row)
    {
      readWrite(row);
      return m_data;
    }
```

Для записи данных должен осуществляться вызов метода readWrite:

```
    void RPGSheet::writeValues(const QJsonObject &
data, int row)
    {
      readWrite(row,data);
```

Приведем реализацию метода readWrite:

```
    void RPGSheet::readWrite(int row, const QJsonObject
&data)
    {
       for (int i = static_cast<int>(RPGEnum::Accessory);
i < static_cast<int>(RPGEnum::_COUNT); ++i) {
          RPGEnum tempEnum = static_cast<RPGEnum>(i);
          switch (tempEnum) {
          case RPGEnum::Latitude:
          case RPGEnum::Longitude:
          case RPGEnum::AB:
          case RPGEnum::DT:
          case RPGEnum::AK:
          case RPGEnum::Parametr3:
          case RPGEnum::Capacity:{
            if(data.isEmpty())
              m_data[enumToString(static_cast
<RPGEnum>(i))]=m_rpgSheet->read(row,i).toDouble();
            else
              m_rpgSheet->write(row,i,data[enumTo-
String(static_cast<RPGEnum>(i))].toDouble());
          }break;
          case RPGEnum::availableTS:
          case RPGEnum::StateTS:
          case RPGEnum::Busing:
          case RPGEnum::Day:{
            if(data.isEmpty())
              m_data[enumTo-
String(static_cast<RPGEnum>(i))] = m_rpgSheet->
read(row,i).toInt();
            else
              m_rpgSheet->write(row,i,data[enumTo-
String(static_cast<RPGEnum>(i))].toInt());
          }break;
          default:{
            if(data.isEmpty())
              m_data[enumTo-
String(static_cast<RPGEnum>(i))] = m_rpgSheet->
read(row,i).toString();
            else
              m_rpgSheet->write(row,i,data[enumTo-
String(static_cast<RPGEnum>(i))].toString());
          }break;
          }
       }
    }
```

Из реализации метода readWrite видно, как осуществляется группировка значений по используемым типам данных. При чтении данных метод readWrite вызывается с передачей ему значения номера считываемой строки, а второй параметр метода используется по умолчанию (константная ссылка – объект QJsonObject, не содержащий какие-либо значения). В ходе работы метода проверяется наличие значений в переменной data; если они отсутствуют, выполняется операция чтения, иначе выполняется операция записи. Представленный пример демонстрирует, что значительно упрощаются масштабирование количества параметров, читаемость кода и его поддержка.

Можно сделать вывод, что данный подход рекомендуется использовать при частых изменениях структуры параметров модели и минимальных либо отсутствующих аналитических проверках вводимых параметров на основе уже введенных (хранящихся значений в файле) данных. Это ограничение обусловлено необходимостью формирования менеджера проверки, в котором будут собираться нужные данные при открытии файла, редактироваться при вводе, удалении или изменении текущих параметров. В ряде случаев проверка может осуществляться по нескольким уже имеющимся значениям параметров или с учетом их агрегации. Все это ведет к усложнению структуры ПО, что сказывается на сложности его поддержки, так как при непосредственной работе с файлами, хранящими исходные значения параметров модели, нет возможности формировать какие-либо запросы (например, SQL и т.д.) к имеющимся данным, поскольку это не предусмотрено стандартными средствами языка программирования и форматом тех файлов, в которых хранятся исходные параметры [8]. В силу этого данные для проверок должны присутствовать в памяти программы и сохранять свою актуальность при изменении значения какого-либо параметра модели, что не всегда удобно в реализации, масштабировании и поддержке данного решения.

Работа с файлом, хранящим исходные параметры, через промежуточную БД

Пример архитектуры ПО ввода и редактирования исходных параметров модели с использованием БД приведен на рисунке 2. БД может создаваться и находиться отдельно от разрабатываемого приложения либо в целевой директории приложения и им же создаваться. Например, SQLite входит в состав *Qt* и по умолчанию подключается добавлением команды *QT + = sql* в pro-файл проекта разрабатываемого приложения.

На рисунке 2 модуль работы с БД отвечает за создание и редактирование (вставить, удалить, изменить) промежуточной БД, а также за организацию запросов, позволяющих проводить проверку вводимых в GUI значений параметров на основе уже имеющихся данных.

Следует отметить, что при таком подходе БД может являться как промежуточным звеном в заполнении и редактировании файлов с исходными параметрами модели, так и источником исходных данных для проведения моделирования. В первом случае можно использовать, как показано на рисунке 2, подключаемые плагины, работающие со своим форматом файлов, которые могут содержать исходные данные. При таком подходе каждый из плагинов при работе с файлом должен обеспечивать:

– считывание хранящихся в файле значений;

– заполнение промежуточной БД считанными значениями из файла в соответствии с ее структурой;

– сохранение (запись) значений параметров, хранящихся в промежуточной БД, в файл.

Работать с данными в БД посредством запросов для их последующего использования при проверке вводимых значений намного проще, чем реализовать подобный механизм самостоятельно или вводить менеджер проверки, в котором необходимые данные будут собираться при открытии файла, редактироваться при вводе, удалении или изменении текущих параметров.

Помимо этого, данный подход позволяет использовать БД как SQL [9], так и NoSQL [10].

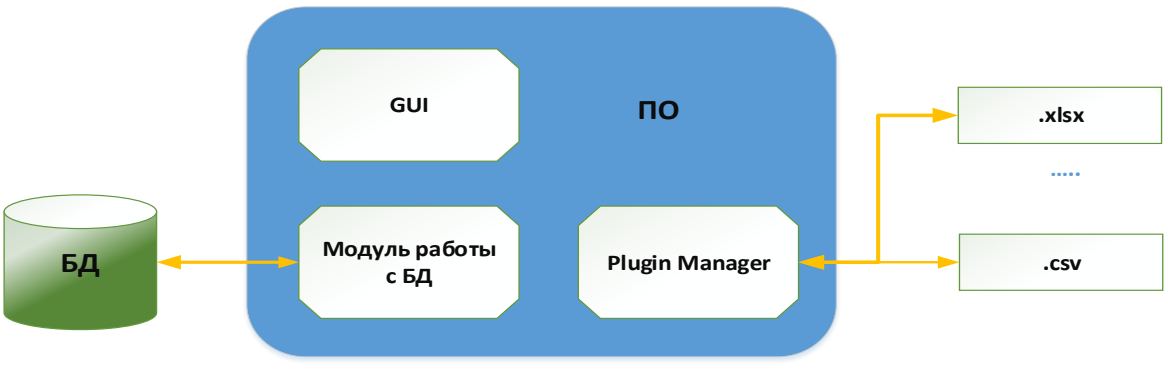

Рис. 2. Пример упрощенной архитектуры ПО

Fig. 2. The simplified software architecture example

В первом случае должны быть заранее определены логические требования к данным, то есть они должны иметь уже устоявшуюся структуру. Получается, что использование реляционных БД на начальных этапах разработки, когда нет понимания относительно количества и структуры параметров, неприемлемо. Этих недостатков лишены БД NoSQL, которые хорошо проявляют себя при нечетких требованиях к данным. Их использование обеспечивает гибкость и масштабируемость модели с поддержкой большого объема данных.

Заключение

Декомпозиция сложного технического процесса является определяющим условием построения его имитационной модели. При этом в зависимости от формата и способа представления исходных данных, а также условий постановки задачи одни и те же этапы могут представляться с различным уровнем детализации. Исходя из описания подходов к разработке ПО ввода исходных параметров с последующим их использованием в имитационных моделях в целях упрощения хранения и редактирования, можно сделать следующие выводы.

Разработка приложения при непосредственной работе с файлом, в котором хранятся исходные параметры, используемые для моделирования, имеет смысл при частом изменении структуры параметров модели и минимальных либо отсутствующих аналитических проверках вводимых параметров на основе уже введенных (хранящихся значений в файле) данных. Также данный подход может применяться на начальных стадиях разработки ПО с последующим переходом к архитектурным решениям с использованием БД.

Разработка приложения с использованием БД более приоритетна, поскольку такой подход позволяет использовать запросы, результаты которых будут применяться при проверке вводимых значений и формировании выходных данных. При этом не рекомендуется на начальных этапах разработки, когда нет понимания относительно количества и структуры параметров, использовать реляционные БД. В таких случаях приоритет отдается БД NoSQL.

Литература

1. Антипова С.А. Обзор методов моделирования и инструментальных средств для исследования логистических процессов материально-технического обеспечения войск (сил) // Логистика. 2019. № 2. С. 28–31.

2. Shaker N., Asteriadis S., Yannakakis G.N., Karpouzis K. Fusing visual and behavioral cues for modeling user experience in games. IEEE Transactions on Cybernetics, 2013, vol. 43, no. 6, pp. 1519–1531. DOI: 10.1109/TCYB.2013.2271738.

3. Девятков В.В. Методология и технология имитационных исследований сложных систем: современное состояние и перспективы развития. М., 2013. 448 с.

4. Гамма Э., Хелм Р., Джонсон Р., Влиссидис Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб, 2016. 366 с.

5. Шлее М. Qt 5.10 Профессиональное программирование на C++. СПб, 2018. 1072 с.

6. Новиков Д.А. Кибернетика: Навигатор. История кибернетики, современное состояние, перспективы развития. М., 2016. 160 с.

7. Crockford D. The application/json media type for JSON. Internet Engineering Task Force, 2006. 10 p. DOI: 10.17487/RFC4627.

8. Ivanov D., Sokolov B., Pavlov A. Optimal distribution (re)planning in a centralized multi-stage network under conditions of ripple effect and structure dynamics. J. Operational Research, 2014, vol. 237, no. 2, pp. 758–770.

9. Чан Ван Фу, Щербаков М.В., Сай Ван Квонг. Грамматика запросов для хранилища разнородных данных в проактивных системах // Программные продукты и системы. 2018. Т. 31. № 4. С. 659–666. DOI: 10.15827/0236-235X.124.659-666.

10. Denium M., Mak G., Long J., Rubio D. NoSQL and BigData. Spring Recipes. Apress Publ., 2014, pp. 549–590. DOI: 10.1007/978-1-4302-5909-1_13.

Software & Systems
DOI: 10.15827/0236-235X.129.013-019

Received 18.07.19
2020, vol. 33, no. 1, pp. 013–019

The development software for input of initial data during simulation

S.A. Chernyshev [1,2], *Ph.D. (Engineering), Senior Researcher, Senior Lecturer, chernyshev.s.a@bk.ru*
S.A. Antipova [1], *Ph.D. (Physics and Mathematics), Senior Researcher, samiraspb11@gmail.com*

[1] *General of the Army A.V. Khrulyov Military Academy of Logistics, St. Petersburg, 199034, Russian Federation*
[2] *St. Petersburg State University of Aerospace Instrumentation, St. Petersburg, 190000, Russian Federation*

Abstract. The widespread adoption of digital technologies gives rise to an increasing flow of structured and unstructured information coming from a large number of different, including disparate and loosely coupled sources of information in all spheres of human activity. Particularly acute is the question of streamlining, purposeful structuring of information and data for their subsequent using in the simulation of complex technical processes.

The article analyzes the approaches to simplify the processes for input and editing a significant array of initial data which necessary for the development of complex simulation. The direct work with text and table format files involves frequent changes in the structure of the model parameters with minimal or no analytical checks of the input parameters based on the already entered data. Working with a file that stores the original parameters through the developed structure of the intermediate data representation allows performing of necessary access to the data stored in the file, but at the same time, it is time-consuming and impractical.

The most successful and versatile decision is to work with a file that stores the original settings through an intermediate database according to the authors opinion. The database can act as an intermediate link in the filling and editing of files with the initial parameters of the model and most of the source data for modeling with this approach.The priority for using in this case are non-relational high-performance NoSQL databases that providing a horizontal scalability. In this paper, the authors noted that a database can act as an intermediate link in filling out and editing files with the initial parameters of the model, and it can be a source of initial data during simulation.

Keywords: simulation, object-oriented programming, database, data format.

References

1. Antipova S.A. The overview of modeling methods and tools for studying logistics processes of troops material support. *Logistics*. 2019, no. 2, pp. 28–31 (in Russ.).

2. Shaker N., Asteriadis S., Yannakakis G.N., Karpouzis K. Fusing visual and behavioral cues for modeling user experience in games. *IEEE Trans. on Cybernetics.* 2013, vol. 43, no. 6, pp. 1519–1531. DOI: 10.1109/TCYB.2013.2271738.

3. Devyatkov V.V. *Methodology and Technology of Complex System Simulation Studies: Current State and Development Prospects.* Moscow, 2013, 448 p.

4. Gamma A., Helm R., Johnson R., Wlissedis J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publ., 1994, 417 p. (Russ. ed.: St. Petersburg, 2016, 366 p.).

5. Shlee M. *Qt 5.10 Professional Programming on C++*. St. Petersburg, 2018, 1072 p.

6. Novikov D.A. *Cybernetics: Navigator. The History of Cybernetics, Current State, Development Prospects*. Moscow, 160 p.

7. Crockford D. The application/json Media Type for JSON. *Internet Engineering Task Force*. 2006, 10 p. DOI: 10.17487/RFC4627.

8. Ivanov D., Sokolov B., Pavlov A. Optimal distribution (re)planning in a centralized multi-stage network under conditions of ripple effect and structure dynamics. *J. Operational Research.* 2014, vol. 237, no. 2, pp. 758–770.

9. Tran Van Phu, Shcherbakov M.V., Sai Van Cuong. Grammar for queries for heterogeneous data storage in proactive systems. *Software & Systems*. 2018, vol. 31, no. 4, pp. 659–666. DOI: 10.15827/0236-235X.124.659-666 (in Russ.).

10. Denium M., Mak G., Long J., Rubio D. NoSQL and BigData. *Spring Recipes.* Apress Publ., 2014, pp. 549–590. DOI: 10.1007/978-1-4302-5909-1_13.

Для цитирования

Чернышев С.А., Антипова С.А. Разработка специального программного обеспечения для ввода исходных данных при имитационном моделировании // Программные продукты и системы. 2020. Т. 33. № 1. С. 013–019. DOI: 10.15827/0236-235X.129.013-019.

For citation

Chernyshev S.A., Antipova S.A. The development software for input of initial data during simulation. *Software & Systems*. 2020, vol. 33, no. 1, pp. 013–019 (in Russ.). DOI: 10.15827/0236-235X.129.013-019.