Информационная система моделирования сложных инженерно-технологических процессов для выбора вариантов решений при проектировании технологических линий

К.Д. Барсегян, Ю.А. Андриенко

Аннотация. Цель настоящий работы – создание методики разработки программного обеспечения для реализации выбора решений при проектировании сложных инженерно-технологических систем. Обсуждается подходы технологии имитационного моделирования (ИМ), выявлению текущего состояния ее применения в инженерной отрасли, сравнительному анализу программного обеспечения (ПО), реализующему технологию ИМ, изучению наиболее успешных практических кейсов, выполненных с использованием проанализированного ПО. Приведен анализ математического аппарата рассмотренного на предыдущем этапе ПО, а также обосновывается необходимость разработки и формируется концепция предполагаемого решения, а также постановку требований к разрабатываемой системе и ее функциональным и нефункциональным возможностям. Описывается архитектура разрабатываемого ПО, представляет обобщенный процесс разработки, а также приводятся результаты тестирования ПО.

Ключевые слова – имитационное моделирование, производственно-технологический процесс, проектирование технологических линий.

I. ВВЕДЕНИЕ

Моделирование инженерно-технологических процессов стало неотъемлемой частью проектирования любой сложной технологической системы. Современным уровнем решений является проектирование цифрового двойника перспективной инженерной системы параллельно, а иногда и упреждающим образом по сравнению не только с созданием объекта «в железе», но и с детальной проектно-конструкторской разработкой. На текущий момент на рынке российского программного обеспечения имеется дефицит импортозамещенных аналогов систем с таким функционалом. Предмет настоящего исследования – программное обеспечение для имитационного моделирования сложных производственно-технологических процессов. Выполнен анализ ПО, реализующего технологию имитационного моделирования в инженерной отрасли с фокусом на успешные кейсы, обоснована разработка импортонезависимого решения.

Предложена концепции решения, разработаны требования к системе. Выполнено проектирование архитектуры продукта. Разработан и протестирован прототип продукта.

Статья получена 7 мая 2024.

Барсегян Карен Давидович, Национальный Исследовательский Ядерный Университет МИФИ, магистрант, gorbushin.volodja@mail.ru

Андриенко Юрий Анатольевич, Национальный Исследовательский Ядерный Университет МИФИ, доцент, yand@outlook.com

II. АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Имитационное моделирование – это распространенная разновидность аналогового моделирования, реализуемого с помощью набора математических инструментальных средств, специальных имитирующих компьютерных программ и технологий программирования, позволяющих посредством процессов-аналогов провести целенаправленное исследование структуры и функций реального сложного процесса в памяти компьютера в режиме «имитации», выполнить оптимизацию некоторых его параметров [1]. Имитационное моделирование осуществляются с помощью специального программного обеспечения, которое включает в себя надлежащий математический аппарат, языковые средства и т.д.

Существует множество программных решений в тех или иных областях, которые удовлетворяют перечисленным ранее требованиям. Например, в области физического и математического моделирования одними из лучших решений являются Simulink (MATLAB), а также SimInTech, в области имитационного моделирования – AnyLogic и т.д. Такие системы позволяют заранее построить модель функционирования сложной инженерно-технологической системы, а также описать сценарии принятия решений этой системы при различных изменениях (в том числе критических) в виде конечного автомата состояний (State Flow Machine) [2]. Пользователь может изменять входные сигналы системы, также менять внутренние условия и следить за реакцией системы и тем, что будет на выходе из нее.

В инженерной отрасли часто встречается понятие модельно-ориентированного проектирования (МОП). МОП – процесс проектирования, основанный на использовании наглядной имитационной модели будущего изделия, которая является основным носителем информации о его концепции, особенностях конструкции и реализации [3]. МОП является эффективным инструментом, упрощающим процесс проектирования системы –имеется возможность собрать прототип реальной системы в виде модели из стандартных функциональных блоков, что является значительным преимуществом перед написанием сложного программного кода. Модельно-ориентированное проектирование по своей сути реализует V-образный цикл разработки. В этом случае можно выделить следующие этапы [4]:

- Определение требований;
- Построение модели;
- Проектирование закона управления и имитации;
- Реализация;
- Валидация и верификация, тестирование.

Графическое интерпретация подхода представлена на Рисунке 1 [5].

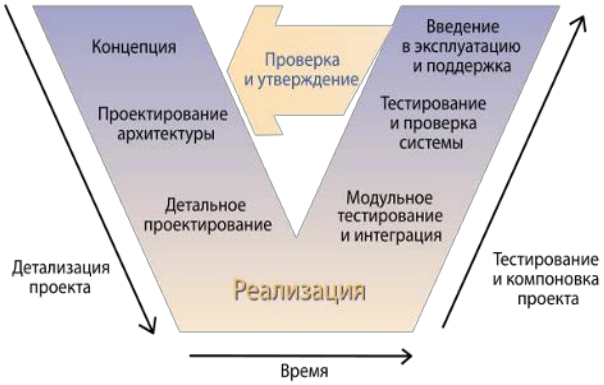

Рис. 1 – V-модель

Основные задачи, которые решает МОП:

- Анализ и проектирование производственных процессов;
- Прогнозирование работы систем и компонентов;
- Оптимизация конструкции и дизайна;
- Управление ресурсами и материалами;
- Прогнозирование рисков и оценка надежности;
- Анализ воздействия изменений;
- Оптимизация процессов тестирования и верификации;

III. СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПО, РЕАЛИЗУЮЩЕГО ПАТТЕРН МОП

Существует большое количество ПО для модельно-ориентированного проектирования, таких как AnyLogic, Simulink, SimInTech, ChemCAD и другие.

Процесс модельно-ориентированного проектирования реализуется инструментом Simulink компании MathWorks. Используется для моделирования и симуляции в авиационной, автомобильной, энергетической и других областях. Позволяет создавать динамические модели систем с использованием блоков и соединений, а также проводить цифровые эксперименты и анализировать их результаты. Обеспечена прямая интеграция с MATLAB, что позволяет использовать язык для написания дополнительного программного кода. Имеется набор функциональных блоков в различных областях – тепловой, электрической, математической, химической и т.д. Реализована визуальная графическая среда для быстрого проектирования модели в виде блоков и стрелок-соединений между ними, что реализует концепцию черного ящика – сигналы текут по соединениям и проходя через функциональные блоки преобразуются в соответствии с логикой, инкапсулированной программным кодом внутри данного конкретного блока. Используются различные решатели (реализация математического аппарата) для реализации дополнительных тонких настроек базовых моделей и т.д. [6]

AnyLogic предоставляет удобный функционал моделирования логистических процессов, функционирования инфраструктуры. Данное ПО реализует парадигмы агентно-ориентированного и дискретно-событийного моделирование и системной динамики. AnyLogic используется при проектировании систем логистики, планировании развития сложных инженерных объектов, оценке взаимного воздействия инфраструктурных проектов и в других областях, требующих сложного анализа взаимосвязанных процессов и систем [7]. Одним из примеров успешного применения является моделирование системы железнодорожного узла в Екатеринбурге. Различие данных на выходе из теоретической модели в AnyLogic и реально собранных экспериментальных данных составила не более 5 %.

SimInTech, разработан в России, позволяет моделировать промышленную автоматизацию – линии сборки и системы управления оборудованием. Также SimInTech позволяет, как и AnyLogic, моделировать и оптимизировать транспортные потоки и логистические решения. При проектировании зданий используется в качестве инструмента для моделирования инженерных систем, в частности силового электроснабжения. Поддерживает дискретное, непрерывное и гибридное моделирование [8].

ChemCAD специализируется на химических процессах и системах. Позволяет проектировать химические установки - дистилляционные колонны, реакторы, теплообменники и т.д. Востребован для объектов нефтепереработки, фармацевтической и пищевой промышленности. Имеет удобный визуальный интерфейс и продвинутый математический аппарат [9].

COMSOL Multyphysics является шведской разработкой, предлагающей инструменты для моделирования физических и инженерных процессов автоматизации. В [10] рассматривается процесс распространения тепла, то есть теплодинамика в современных керамических материалах. Программное обеспечение Comsol Multyphysics подтвердило способность предоставлять надежные результаты при численном моделировании фундаментальных физических процессов.

В статье [11] рассматривается платформа для разработки и тестирования GPSприемника на базе LabVIEW с периферийными устройствами DSP.

Приведем сравнительную таблицу основных ПО для моделирования в инженерной отрасли (Таблица 1).

Таблица 1 – Сравнительный анализ ПО

Продукт	**Simulink**	**SimInTech**	**AnyLogic**	**LabView**	**COMSOL Multyphys-ics**	**ChemCAD**
Языки программирования	C, C++, CUDA, PLC, Verilog, VHDL, MATLAB	Delphi, C, C++, SimInTech built-in language	Java	G, LabView built-in language	COMSOL script builtin language (основан на MATLAB)	Информация ограничена (скорее всего C, C++)
Типы реализуемого моделирования	Непрерывное, дискретное, гибридное	Непрерывное, дискретное, гибридное	Дискретно- событийное, гибридное, агентно-ориентированное	Непрерывное, дискретное, гибридное	Непрерывное, дискретное, гибридное	Непрерывное
Страна-разработчик	США	Россия	США, Россия (права проданы)	США	Швеция	США
Области моделирования	Электроника и электротехника, аэродинамика, гидродинамика, механика и динамика, теплотехника и тепловые процессы, биомедицинская инженерия, коммуникационные системы,	Промышленная автоматизация, электроэнергетика, управление техническими системами, транспортные системы, экологическое моделирование, системы управления зданиями	Логистика и транспорт, здравоохранение, финансы и экономика, городская инфраструктура и планирование, энергетика и экология	Измерения и автоматизация, системы управления и регулирования, сбор и обработка данных, обработка сигналов	Теплопередача и теплообмен, электричество и электротехника, механика и динамика, акустика и звукопередача, химические процессы, гидродинамика, аэродинамика, системы управления и оптимизация	Химическая промышленность, нефтепереработка и нефтехимия, фармацевтика, пищевая промышленность, энергетика и экология

IV. ПОТРЕБНОСТЬ РАЗРАБОТКИ РОССИЙСКИХ СИСТЕМ ИММИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Согласно указу Президента Российской Федерации от 30.03.2022 № 166, с 01.2025 г. будет введен запрет на использование иностранного ПО органами государственной власти и заказчиками на объектах критической информационной инфраструктуры [12]. При этом уже с начала 2022 г. многие российские (как частные, так и государственные) компании совершают переход на отечественные разработки, входящие в Единый реестр российского программного обеспечения.

Несмотря на достаточно обширный рынок программного обеспечения, реализующего технологию имитационного моделирования, его значительная часть представлена зарубежными компаниями и разработками. При проведении сравнительного анализа ПО не было выявлено российских разработок из Росреестра, которые бы могли в полной мере заменить функционал Simulink или Anylogic. В связи с этим полный отказ от иностранного программного обеспечения в данной сфере не представляется возможным.

Одновременно с этим продолжение использования зарубежного ПО может повлечь за собой ряд проблем, включая:

- Отсутствие возможности своевременного обновления ПО;
- Отказ в предоставлении технической поддержки со стороны разработчика;
- Неконтролируемое отключение важных функций программы;
- Потеря оплаченной лицензии без возврата денежных средств;
- Потеря доступа к удаленным серверам и базам данных;
- Угроза нарушения безопасности и потери данных [12].

В связи с этим критически важной задачей является разработка импортозамещенного программного решения, реализующего технологию имитационного моделирования, которое бы стало независимым от иностранных проприетарных технологий отечественным аналогом и могло безопасно и на законных основаниях применяться в деятельности российских инженерно-технологических частных и государственных компаний.

V. ТРЕБОВАНИЯ К СИСТЕМЕ ИММИТАЦИОННОГО МОДЕЛИРОВАНИЯ

К разрабатываемой системе выдвигаются функциональные (Таблица 2) и нефункциональные требования (Таблица 3).

Таблица 2 – Функциональные требования к системе

№	Функциональное требование
1	В системе должно быть реализовано управление функциональными блоками и редактирование их параметров
2	Система должна выдавать результаты в виде результирующих показателей по окончанию процесса моделирования
3	В системе должна быть реализована проверка синтаксиса (нотации) элементов модели
4	Система должна позволять использовать типовые шаблоны процессов
5	Система должна предъявлять рекомендации по изменению процесса на основе результирующих показателей
6	Система не должна допускать модели с ошибками к запуску

Таблица 3 – Нефункциональные требования к системе

№	Нефункциональное требование
1	Система должна быть импортонезависимой и не содержать иностранных проприетарных составляющих частей
2	Интерфейс системы должен быть переключаемым на русский и английский языки
3	Система должна обеспечивать одновременное подключение 250 пользователей
4	Время исполнения одной модели не должно составлять более 30 минут
5	Моделирование в системе должно осуществляться без использования программирования
6	Система должна быть масштабируема для применения в других отделах / смежных сферах деятельности

VI. АРХИТЕКТУРА ПРОГРАМНОГО РЕШЕНИЯ

В первую очередь для создания ПО была продумана детальная архитектура. В реализации ПО, работающих по принципу блок-схем и соединений между ними есть несколько ключевых моментов, которые стоит учесть. Списки блоков и соединений между ними должны храниться в какой-либо коллекции, поэтому может быть два варианта:

- Добавление или удаление блоков и соединений между ними в графической среде немедленно отражается на коллекции, в которой будут хранится блоки и соединения;
- Конечная коллекция блоков и соединений передается графической средой с помощью готового объекта (какого-либо текстового описания модели) в ядро программы.

Первый вариант предоставляет более жесткую связь графической среды и ядра программы, т.е. frontend-а и backend-а, поскольку должен быть немедленный отклик на изменения.

Второй вариант предоставляет большую гибкость в том смысле, что ядро программы в таком случае является независимым backend сервисом, к которому можно присоединить собственно-разработанный frontend-сервис и с помощью указанных API-контрактов.

VI.I СПОСОБ ОПИСАНИЯ МОДЕЛИ

В качестве объекта конечного описания модели в разрабатываемой ПО является текстовое описание ее блоков и соединений между ними в формате JSON, то есть фактически описание графа.

Для начала требуется договориться, что из себя представляют блок и соединение в смысле данных. Введем понятие порт, как объект, который привязан к отдельно взятому блоку и является его атрибутом. Порты естественным образом разделяются на входные и выходные – каждое соединение между двумя блоками начинается в выходном порте одного блока и заканчивается во входном порте другого. Соответственно, соединение представляет собой линию, по которой «течет» сигнал – сигнал, который один блок вычисляет, передается через его выходной порт по соединению во входной порт другого блока, где далее обрабатывается этим блоком. Таким образом, описание модели содержит описание блоков в виде списка, каждый элемент которого является отдельным блоком и состоит из описания следующих элементов:

- Имя блока – поле Name;
- Тип блока – поле Type;
- Уникальный идентификатор блока – поле Id;
- Список входных портов, каждый элемент которого содержит уникальный идентификатор входного порта – поле InputPorts;
- Список выходных портов, каждый элемент которого содержит уникальный идентификатор выходного порта – поле OutputPorts;
- Параметры блока в виде словаря – поле Params.

А также описание соединений – каждое из них состоит из следующих элементов:

- Уникальный идентификатор соединения – поле Id;
- Уникальный идентификатор выходного порта блока – поле InportId; • Уникальный идентификатор входного порта блока – поле OutportId; Описание также должно содержать условия симуляции:
- Момент начала симуляции – поле TStart;
- Момент конца симуляции – поле Tend;
- Шаг симуляции – поле TStep.

Разработанное ПО имеет контракт на получение в точку входа в программу описание выше. При входе в про-

грамму JSON, полученный на вход анализируется и переводится в программное представление с помощью специальных инструментов. Модель из JSON внутри программы хранится в объекте той структуры, на которую он был «натянут». Каждый объект, который описан в JSON также имеет свое внутреннее программное представление, эти объекты являются атрибутами программного представления всей модели, к которым через нее можно получить доступ.

VI.II ОПИСАНИЕ КЛАССОВ И ИНТЕРФЕЙСОВ

Архитектура содержит в себе описание базовых типов в виде 5 интерфейсов и 10 классов. Как мы отметили ранее, описание модели имеет в себе такие понятия как сама модель, блок, порт, соединение и ее настройки.

В целях разделения ответственности было также введено понятие метаинформации о блоке и модели. Метаинформация блока представляет собой информацию вспомогательного характера, состоящую из информации о входных и выходных портах, уникального идентификатор блока и его имени. Для модели в целом метаинформация представляет собой информацию о хранящихся в ней блоках, соединениях, всех выходных и входных портах в модели, настройках симуляции – начала, конца и шага симуляции, а также последовательности вызовов – понятии, которое будет введено дальше.

Начнем рассмотрение типов с интерфейсов. Пять интерфейсов предоставляют базовые методы для работы с объектами базовых классов. Рассмотрим их:

A. ИНТЕРФЕЙС IModelMeta

Интерфейс IModelMeta предоставляет методы для работы с метаинформацией о модели:

- Метод AddBlock принимает в себя параметры id и block и позволяет добавить в список блоков очередной блок;
- Метод RemoveBlock принимает в себя уникальный идентификатор блока, который нужно удалить из общего списка;
- Метод AddConnection принимает в себя объект типа Connection, представляющего из себя базовый тип соединения, который нужно добавить в общий список соединений;
- Метод RemoveConnection принимает в себя уникальный идентификатор соединения, который нужно удалить из общего списка соединений.

B. ИНТЕРФЕЙС IModel

Интерфейс IModel предоставляет следующие методы:

- Метод Step, который принимает текущий момент времени t и продвигает исполнение модели на шаг вперед;
- Метод Simulate, который ничего не принимает в качестве аргументов и при вызове начинает симуляцию и с течением времени вызывает Step на каждом шаге, пока текущий момент времени не станет равным времени симуляции, выставленном в настройках.

C. ИНТЕРФЕЙС IBlockMeta

Интерфейс IBlockMeta предоставляет следующие методы:

- Метод AddInport, принимающий в качестве аргументов объект базового типа Port, который представляет из себя входной порт блока и добавляет в общий список входных портов блока;
- Метод AddOutport, принимающий в качестве аргументов объект базового типа Port, который представляет из себя выходной порт блока и добавляет в общий список выходных портов блока;
- Метод RemoveInport, принимающий в себя уникальный идентификатор входного порта и удаляет из общего списка входных портов;
- Метод RemoveOutport, принимающий в себя уникальный идентификатор выходного порта и удаляет из общего списка выходных портов.

D. ИНТЕРФЕЙС IBlock

Интерфейс IBlock предоставляет следующие методы:

- Метод Step, принимающий в качестве аргумента текущий момент времени t, и рассчитывающий выходные значения блока;
- Метод Update, принимающий в качестве аргумента текущий момент времени t, и обновляющий внутреннее состояние блока. Здесь стоит отметить, что у блока может быть внутреннее состояние, которое обновляется с каждым шагом и на основе которого рассчитываются выходные значения;
- Метод GetLoggedValues, ничего не принимающий на вход и возвращающий массив выходных значений. Является своеобразным логирующим методом с помощью которого можно получать выходные значения какого-либо блока.

E. ИНТЕРФЕЙС IConnection

Интерфейс IConnection предоставляет следующий единственный метод

- Метод PropagateValue, который принимает на вход список всех входных и выходных портов модели, и распространяющий сигналы по заданному соединению, т.е. от выходного порта одного блока до входного порта другого блока, заданных в атрибутах конкретного соединения.

Перейдем к описанию базовых классов. Начнем с описания классов (структур), на которые «натягиваются» объекты из JSON, полученного на входе программы. Имеются следующие четыре структуры.

A. КЛАСС JsonPort

Класс JsonPort – описывает структуру порта, приходящего из JSON, содержит в себе единственное поле:

- Поле Id – уникальный идентификатор порта.

B. КЛАСС JsonConnection

Класс JsonConnection описывает структуру соединения, приходящего из JSON, содержит в себе следующие поля:

- Поле Id – уникальный идентификатор соединения;
- Поле InportId – уникальный идентификатор выходного порта;
- Поле OutportId – уникальный идентификатор входного порта.

C. КЛАСС JsonBlock

Класс JsonBlock описывает структуру блока, приходящего из JSON, содержит в себе следующие поля (Рисунок 16):

- Поле Id – уникальный идентификатор блока;

- Поле Name – уникальное имя блока;
- Поле Type – тип блока;
- Поле Inports – список входных портов блока;
- Поле Outports – список выходных портов блока;
- Поле Params – словарь, состоящий из параметров конкретного блока (варьируется для каждого типа блока).

D. КЛАСС JsonModel

Класс JsonModel описывает структуру модели, приходящей из JSON, содержит в себе следующие поля:

- Поле TStart – хранит время начала симуляции;
- Поле Tend – хранит время конца симуляции;
- Поле TStep – хранит шаг симуляции;
- Поле Blocks – хранит в себе список блоков;
- Поле Connections – хранит в себе список соединений между блоками.

Далее рассмотрим базовые типы, в которые переводятся промежуточные представления объектов (JSON-типы).

A. КЛАСС Port

Класс Port описывает структуру порта. Представление в виде JsonPort переводится в данный тип. Класс содержит следующие поля и методы:

- Поле Id – уникальный идентификатор порта, совпадает с Id из промежуточного представления;
- Поле Value – значение, хранящееся на порте. Для входного порта блока это входное значение, для выходного порта – выходное значение;
- Статический метод CreatePort, принимающий в качестве аргумента промежуточное представление порта (объект типа JsonPort). Метод создает объект типа Port на основе объекта типа JsonPort, фактически являющийся фабричным методом.

B. КЛАСС Connection

Класс Connection описывает структуру соединения. Представление в виде JsonConnection переводится в данный тип. Класс наследует интерфейс IConnection, содержит следующие поля и методы:

- Поле Id – уникальный идентификатор соединения, совпадает с Id из промежуточного представления;
- Поле InportId – уникальный идентификатор выходного порта, совпадает с
- InportId из промежуточного представления;
- Поле OutportId – уникальный идентификатор входного порта, совпадает с OutportId из промежуточного представления;
- А также методы, наследованные от интерфейса IConnection.

C. КЛАСС BlockMeta

Класс BlockMeta описывает структуру, хранящую метаинформацию о блоке. Данный класс наследует интерфейс IBlockMeta и содержит следующие поля и методы (Рисунок 16):

- Поле Id – уникальный идентификатор блока, совпадает с Id из промежуточного представления блока;
- Name – уникальное имя блока, совпадает с именем блока из промежуточного представления;
- Поле Inports – словарь, ключом которого является уникальный идентификатор входного порта, а значением – входной порт;
- Поле Outports – словарь, ключом которого является уникальный идентификатор выходного порта, а значением – выходной порт;

А также методы, наследованные от интерфейса IBlockMeta.

D. КЛАСС Block

Класс Block описывает структуру, хранящую информацию о входных и выходных значениях блока, а также в виде внедренной зависимости хранит в себе объект типа IBlockMeta. Класс наследует интерфейс IBlock. Рассмотрим подробнее поля и методы, содержащиеся в классе:

- Поле BlockMeta – внедренная зависимость метаинформации о блоке;
- Поле InputValues – хранит входные значения на блоке в виде массива, так как блок может иметь больше одного входного порта;
- Поле OutputValues – хранит выходные значения на блоке в виде массива, так как блок может иметь больше одного выходного порта;
- Метод WriteValues, ничего не принимающий в качестве аргументов. Вызывается внутри метода Step, чтобы записать рассчитанные значения на выходные порты;
- Метод ReadValues, ничего не принимающий в качестве аргументов. Вызывается внутри метода Update, чтобы записать входные значения на текущем шаге;
- А также методы, наследованные от интерфейса IBlock.

E. КЛАСС ModelMeta

Класс ModelMeta описывает структуру, хранящую метаинформацию о модели, наследует интерфейс IModelMeta. Рассмотрим подробнее поля и методы класса:

- TStart – время начала симуляции;
- Поле Tend – время конца симуляции;
- Поле TStep – шаг симуляции;
- Поле Blocks – словарь, ключом которого является уникальный идентификатор блока, а значением – блок;
- Поле Connections – словарь, ключом которого является уникальный идентификатор соединения, а значением – соединение;
- Поле Inports – словарь, ключом которого является уникальный идентификатор входного порта, а значением – входной порт;
- Поле Outports – словарь, ключом которого является уникальный идентификатор выходного порта, а значение – выходной порт;
- Поле CallTrace – словарь, ключом которого является блок, а значением – список соединений, относящихся к данному блоку. Это поле представляет из себя информацию о том, в каком порядке следует вызывать методы Update и Step каждого блока и распространять значения по соединениям с помощью метода PropagateValue, связанным с этим блоком. Фактически списки ключей и значений являются направленными и задают этот порядок. Позже это поле будет рассмотрено подробнее при изучении алгоритма исполнения модели.

- Статический метод CreateModel, принимающий в качестве аргумента промежуточное представление модели в виде объекта типа JsonModel. Является фабричным методом, который создает объект типа ModelMeta;
- Метод FillCallTrace, не принимающий аргументов на вход и наполняющий на основе внутреннего состояния модели поле CallTrace, описанное выше.

F. КЛАСС Model

Класс Model описывает структуру конечного представления модели. Класс наследует интерфейс IModel. Содержит следующие поля и методы:

- ModelMeta – внедренная зависимость, хранящая метаинформацию о
- модели (объект типа IModelMeta);

А также методы, наследованные от интерфейса IModel.

VII. РЕАЛИЗАЦИЯ ПРОДУКТА НА УРОВНЕ MVP

Написание любого блока для наполнения библиотеки содержит в себе всего один этап разработки – написание класса блока, унаследованного от базового типа Block. Написание включает в себя:

- Описание полей-параметров блока;
- Описание внутреннего состояние блока, которого может и не быть;
- Наследование метода Step, реализующего расчет выходных значений блока;
- Наследование метода Update, реализующего обновление внутреннего состояния блока.

Рассмотрим на конкретном примере реализацию блока без внутреннего состояния – блока SineWave (аналога из Simulink), выдающего синусоидальный сигнал на выходе. Полями-параметрами этого блока являются четыре величины – амплитуда (*Amplitude*), фаза (*Phase*), частота (*Frequency*) и начальное смещение по оси Y (*Bias*). Блок не содержит по логике никаких параметров, которые являлись бы внутренним состоянием блока, поэтому метод Update для данного блока не будет содержать никакой реализации. Метод Step рассчитывает на выходе значение по следующей формуле:

$$y = Amplitude * sin(Frequency * Time + Phase) + Bias \quad (1)$$

Примером блока с внутренним состоянием является интегратор с дискретным временем. Такой блок в простейшем случае содержит единственный параметр, описывающий внутреннее состояние – *state*. Метод Update в себе будет содержать именно обновление этого внутреннего состояния по следующей формуле:

$$state = state + step * input \quad (2)$$

где *step* – шаг интегрирования, input - входное значение блока. Метод Step в свою очередь не будет содержать логики и будет просто возвращать внутреннее состояние *state*.

Рассмотрим алгоритм работы программы в виде конвейера обработок (Рисунок 2).

Точкой входа в программу является JSON-описание модели. Когда программа принимает его на вход, происходит процесс натяжения этого JSON на структуры JsonModel, JsonBlock, JsonConnection и JsonPort. Здесь, как описывалось ранее, происходит первый конвейерный процесс обработки, текстовое описание модели переводится в промежуточное программное представление для последующей обработки.

Вторым этапом создается следующее программное представление модели – объект типа ModelMeta. Для этого вызывается статический метод CreateModel, в который передается промежуточное представление. В свою очередь в данном метода происходит обработка полей, объявленных в JsonModel, и их надлежащая обработка для наполнения собственного представления модели.

Третьим и последним этапом конвейерной обработки является создание объекта типа Model. Для этого требуется только передать в конструктор класса ранее созданный объект типа ModelMeta. Для запуска симуляции остается только вызвать метод Simulate от объекта типа Model. Рассмотрим этапы процесса симуляции, а конкретнее одного его цикла:

1. В начале каждого цикла обновляется время симуляции. К предыдущему моменту времени прибавляется шаг симуляции, заданный в настройках, и получается текущий новый момент симуляции.
2. Алгоритм симуляции последовательно проходится по всем блокам, хранящимся в поле CallTrace, и вызывает от каждого блока сначала метод Update, и только потом метод Step.
3. Затем, от всех соединений, хранящихся в значениях словаря CallTrace по ключу текущего блока, вызывается метод PropagateValue для обновления значения на входных портах следующих за данным блоком блоков.
4. Если требуется, значения логированных сигналов сохраняются в список для дальнейшего вывода.
5. Данный цикл повторяется, пока текущий момент времени не превысит время конца симуляции.

Рис. 2. Цикл расчета

Для разработки был выбран язык C#. Фактически после создания архитектуры ПО остается реализовать только конкретные методы, описанные в ней. Здесь есть несколько важных моментов, которые стоит описать:

1. Реализация метода FillCallTrace, который создает последовательность вызовов блоков и соединений;
2. Реализация метода CreateModel, переводящего промежуточное представление JsonModel в представление ModelMeta;
3. Реализация метода Step и Simulate класса Model;
4. Пример написания нового блока.

Рассмотрим каждый из этих пунктов отдельно.

A. РЕАЛИЗАЦИЯ МЕТОДА FillCallTrace

Любая модель представляет из себя некий направленный граф, где узлами являются блоки, а ребрами направленные соединения между блоками. Важно также ввести понятие источников и приемников. Источники –

блоки, у которых присутствуют выходные порты, но отсутствуют входные порты, т.е. такие блоки только выдают сигнал. Приемники – блоки, у которых присутствуют входные порты, но отсутствуют выходные порты, т.е. такие блоки только принимают сигнал. Фактически, источники являются корнями графа, а приемники – листьями графа.

Для упрощения работы программы ранее было введено понятие стека вызовов CallTrace. Если программа каждый шаг симуляции будет анализировать граф и вызывать в правильном порядке блоки и соединения, это будет занимать неразумно долгое время. Вместо этого можно один раз заранее проанализировать граф и составить тот самый стек вызовов. А в дальнейшем просто на каждом шаге симуляции идти вдоль стека вызовов и совершать вызовы, так как они уже заранее упорядочены.

Рассмотрим пример модели, которая показывает с какими проблемами можно столкнуться, а заодно рассмотрим на ней алгоритм работы метода FillCallTrace, который создает CallTrace (Рисунок 3).

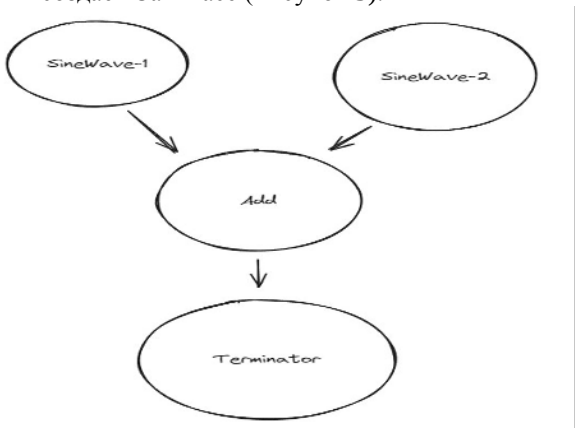

Рис. 3. Простейшая модель в виде графа

Каждый блок может иметь несколько входных и выходных портов, поэтому на графе в блок может входить и выходить несколько стрелок. Один шаг симуляции происходит при одном и том же заранее определенном в начале шага моменте времени, поэтому каждый блок должен ожидать пока на все его входные порты не будут записаны новые значения от предыдущих блоков. Таким образом, если у блока два входных порта, то он ожидает вызова двух соединений для того, чтобы на два его входных порта были записаны новые значения. На примере выше таким блоком является блок Add (аналог из Simulink) – сумматор, который суммирует все приходящие в него значения. Блоки SineWave-1 и SineWave-2 (аналог SineWave из Simulink) являются источниками синусоидального сигнала, а блок Terminator (аналог из Simulink) является приемником сигнала – никак не обрабатывает сигнал, и является просто блоком для замыкания графа.

Обработку графа и создание стека CallTrace следует начинать от источников сигнала. Полный алгоритм обработки графа можно увидеть на Рисунке Б1 в Приложении Б. Код алгоритма находится в Приложении В, пункте В1. Опишем более общими словами шаги алгоритма:

1. Начинаем с источников и их соединений и заводим текущий уровень блоков. Будем называть текущим уровнем блоков список блоков, который обрабатывается на текущем этапе. Добавляем их в CallTrace.
2. Заводим список связанных блоков. Находим все блоки, связанные с блоками текущего уровня, и проверяем если у них больше одного входного порта, то добавляем их в специальный список. Если блок уже есть в этом списке, то помечаем, что на очередной его входной порт пришло новое значение, при этом если оказалось, что все входные порты обработаны, то удаляем из специального списка и вносим в список связанных блоков. В противном случае добавляем блок в список связанных блоков. Таким образом, осуществляется ожидание, что все входные порты блока обработаны, и его можно добавить в нужное место стека вызовов.
3. Очищаем текущий уровень и добавляем туда список связанных блоков и снова переходим к пункту 1, пока текущий уровень не станет пустым.

B. РЕАЛИЗАЦИЯ МЕТОДА CreateModel

Важным шагом является перевод промежуточного представления JsonModel в представление ModelMeta. Полный код этого алгоритма находится в Приложении В, пункте В1. Здесь возникает проблема, что требуется создавать объекты типов, определенных в библиотеке блоков во время алгоритма перевода, поскольку типы блоков, которые нужно создавать определены в текстовом виде в поле Type класса JsonBlock. Но эту проблему можно решить с помощью имеющегося в C# инструмента для позднего связывания, который предлагается пакетом System.Activator. С помощью метода GetType сначала получаем программное представление типа, указанного в текстовом виде в поле Type. А также требуется передать аргументы конструктора отдельным списком типа object. Далее создаем объект нужного типа с помощью метода CreateInstance, который принимает тип блока и аргументы его конструктора.

Сам по себе метод CreateModel получает данные из объекта типа JsonModel и наполняет свои внутренние свойства нужным образом на основе данных из JsonModel.

C. МЕТОДЫ Step и Simulate класса Model

Рассмотрим методы, отвечающие за процесс симуляции. В Приложении В, пункте В2 представлен программный код, реализующий методы Step и Simulate.

Алгоритм шага достаточно прост и состоит всего из трех шагов:

1. Вызвать Update блока;
2. Вызвать Step блока;
3. Вызвать PropagateValue от всех его выходных соединений;

Эти три шага повторяются для каждого блока в списке CallTrace. При этом текущий момент времени передается в метод в качестве аргумента, а управляет ходом времени метод Simulate, который инкрементирует момент времени на шаг симуляции и вызывает метод Step после каждого инкремента.

D. ПРИМЕР РЕАЛИЗАЦИИ НОВОГО БЛОКА В БИБЛИОТЕКЕ

Код экземпляра можно посмотреть в Приложении В, пункте В3. Написание нового блока можно разделить на следующие шаги:

1. Объявление нового класса, наследующего класс Block;
2. Объявление внутренних параметров блока, в примере с SineWave такими являются Amplitude, Phase, Frequency, Bias;
3. Объявление конструктора класса, который принимает объект типа BlockMeta, где хранится метаинформация о блоке, а также словарь, где ключом являются названия параметров, соответствующих тем, что описаны в пункте 2, а ключом их значения.
4. Реализация метода Step, включает описание логики расчета выходных значений блока. Для данного примера – Формула 1. В конце каждого Step должна быть инструкция (если блок не является приемником) WriteValues (), которая запи-
5. сывает рассчитанные значения на выходные порты
6. Реализация метода Update, включает описание логики обновления внутренних состояний, если таковые есть. Для данного примера блок не имеет внутренних состояний, поэтому метод пустой. В начале этого метода всегда должна быть инструкция (если это не источник) ReadValues (), которая считывает пришедшие значения со входных портов.

VIII. ТЕСТИРОВАНИЕ И ДЕМОНСТРАЦИЯ

В качестве тестовой модели рассмотрим чайник – простейший с точки зрения бытового пользования прибор. Будем сравнивать результаты с результатами из Simulink. Изначально для построения модели определим ее математическое описание в виде уравнений.

Первое, что требуется определить – зависимость температуры с течением времени. Отметим некоторые важные моменты:

1. Будем считать, что на чайник подается линейно возрастающее со временем напряжение.
2. Пар начинает образовываться только при достижении жидкостью точки закипания.

Первое, что требуется определить – зависимость напряжения:

$$U = K_u t \tag{3}$$

где U – напряжение в момент времени t с начала включения чайника, Ku – коэффициент линейного прироста напряжение. Далее определим теплоту, выделяемую нагревательным элементом:

$$Q = \frac{U^2}{R}\, t \tag{4}$$

где Q – выделяемая теплота, R – сопротивление нагревательного элемента.

Далее определим главную зависимость – изменение температуры жидкости:

$$T = \begin{cases} T_0 + \frac{Q\,\eta}{C\,m_B}\ ℃,\ T < 100℃ \\ 100\ ℃,\ T = 100℃ \end{cases} \tag{5}$$

где T_0 – начальная температура жидкости, η – коэффициент полезного действия прибора, C – удельная теплота нагревания жидкости, $m_в$ – масса жидкости.

Также определим давление, накапливаемое в пустом пространстве чайника:

$$P = P_0 + \gamma \frac{m_п}{V} \tag{6}$$

где P_0 – атмосферное давление, $m_п$ – масса образуемого пара, V – объем пустого пространства в чайнике, γ – коэффициент зависимости давления от плотности пара. Закон образования массы пара со временем:

$$m_п = \begin{cases} \frac{Q\,\eta}{l}\ ,\ T \geq 100℃ \\ 0\ ,\ T < 100℃ \end{cases} \tag{7}$$

где l – удельная теплота парообразования жидкости. Также определим логическую величину, определяющую «свист» чайника:

$$ready = \begin{cases} 1\ , P \geq\ \alpha\ P_0 \\ 0\ , P <\ \alpha\ P_0 \end{cases} \tag{8}$$

где a – коэффициент, определяющий границу давления, при которой чайник начнет свистеть.

Модель в простейшем виде включает всего 7 типов блоков:

- Constant – блок, выдающий константное значение на каждом шаге моделирования, служит для определения константных значений и начальных условий;
- Clock – блок, выдающий текущее время моделирования;
- Product – блок, перемножающий или делящий (в зависимости от определенного арифметического знака для каждого входного значения) входные значения;
- Add – блок, суммирующий или вычитающий (в зависимости от определенного арифметического знака для каждого входного значения) входные значения; • Switch – блок, который выводит один из двух сигналов, поданных на вход в зависимости от третьего, управляющего сигнала, который проверяется на некоторое условие. Является своеобразным логическим вентилем в виде транзистора;
- Square – блок, который возводит в квадрат входной сигнал;
- Scope – блок, который строит график на основе входного сигнала.

Результаты расчета приведенной модели описанным методом иллюстративно приведены на рисунках 4 – 9.

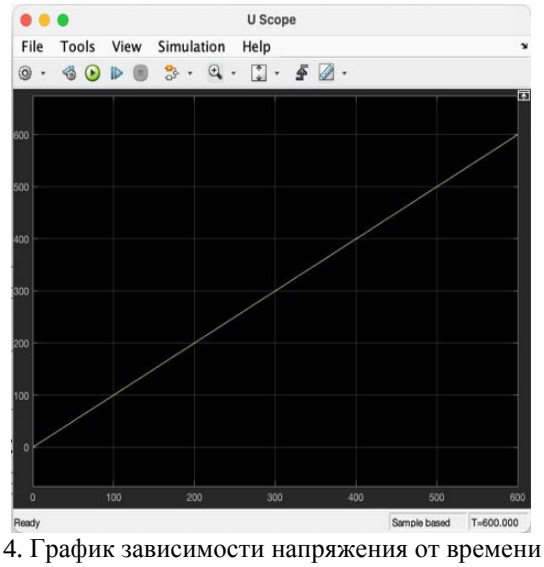

Рис. 4. График зависимости напряжения от времени

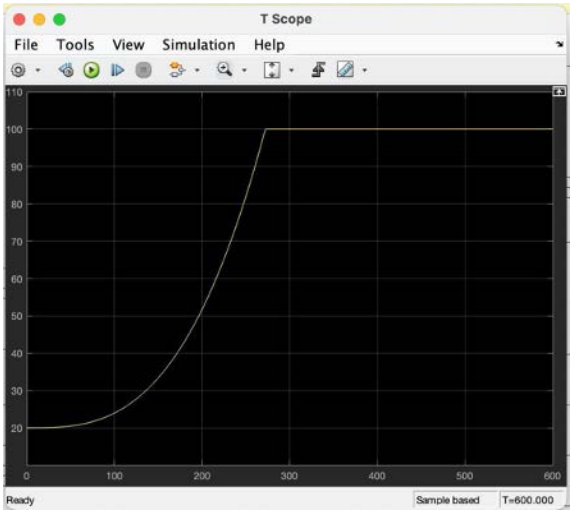

Рис. 5. График зависимости температуры воды от времени.

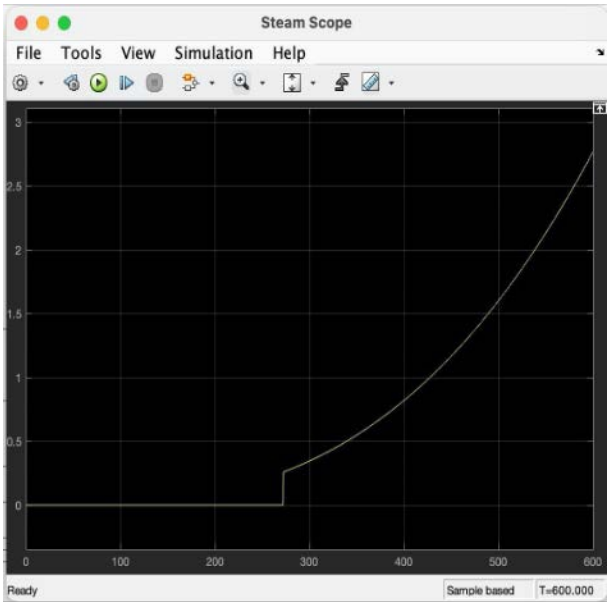

Рис. 6. График зависимости массы пара от времени.

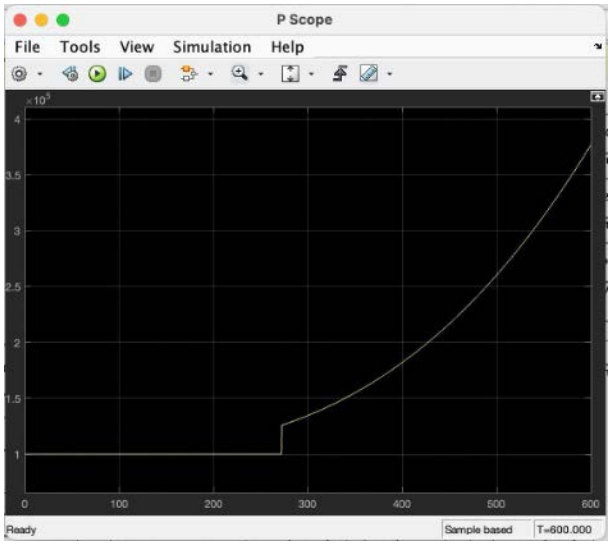

Рис. 7. График зависимости давления на свисток чайника от времени.

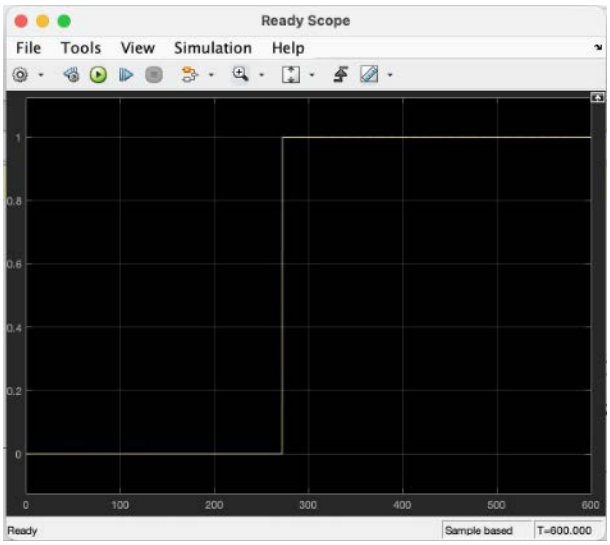

Рис. 8. График зависимости логической величины, соответствующей срабатыванию свистка

IX. ЗАКЛЮЧЕНИЕ

В результате выполнения представленной работы были решены следующие задачи:

- Изучена технология имитационного моделирования;
- Изучено применение технологии моделирования в инженерной отрасли;
- Произведен сравнительный анализ ПО, реализующего технологию имитационного моделирования в инженерной отрасли;
- Произведен анализ успешно реализованных кейсов с использованием данных ПО;
- Приведено обоснование необходимости разработки импортонезависимого решения;
- Предложена концепция решения;
- Выполнена постановка требований к разрабатываемой системе;
- Спроектирована архитектура ПО;
- Разработано ПО, описан процесс его разработки;
- Произведено тестирование разработанного ПО.

Предложенная методика планируется нами к применению для широкого круга приложений, таких как моделирование систем обращения с ЖРС и ЖРО АЭС, разработка технологических режимов целлюлозно-бумажного производства и разработка АСУТП аквабиотехнологического производства на базе прогноза последствий технологических решений.

БЛАГОДАРНОСТИ

Авторы выражают благодарность Высшей инжиниринговой школе НИЯУ МИФИ за помощь в возможности опубликовать результаты выполненной работы.

БИБЛИОГРАФИЯ

[1] Н.А. Поляков, История имитационного моделирования, стр. 1, 2012

[2] H. Bourbouh, P. L. Garoche, C. Garion, Automated analysis of Stateflow models. EPiC Series in Computing, v. 46, p. 144-161, 2017.

[3] Ю.З. Талукдер, Модельно-ориентированное проектирование систем автоматического управления в инженерном образовании, с. 3, 2013.
[4] Н.П. Деменков, Модельно-ориентированное проектирование систем управления, с. 2.
[5] Интернет-ресурс «Википедия: V-Model». [Режим доступа]: https://ru.wikipedia.org/wiki/V-Model.
[6] Интернет-ресурс «Экспонента: Simulink». [Режим доступа]: https://exponenta.ru/simulink.
[7] Интернет-ресурс «Anylogic». [Режим доступа]: https://www.anylogic.com.
[8] Интернет-ресурс «SimInTech: Справочная система». [Режим доступа]: https://help.simintech.ru/#o_simintech/browsers.html.
[9] Интернет-ресурс «Chempute software: ChemCAD». [Режим доступа]: https://chempute.com/chemcad.html.
[10] Mohammad Vajdi, Farhad Sadegh Moghanlou, Fariborz Sharifianjazi, Mehdi Shahedi Asl, Mohammadreza Shokouhimehr, A Review on the Comsol Multyphysics studies of heat transfer in advanced ceramics, Journal of Composites and Compounds, p. 35-43, 2020. [Режим доступа]: https://www.researchgate.net/publication/340437687_A_review_on_the_Comsol_Multiphysics_studies_of_heat_transfer_in_advanced_ceramics.
[11] Arpine Soghoyan and David Akopyan, A LabVIEW-Based GPS Receiver Development and Testing Platform with DSP Peripherals: Case study with C6713 DSK, Journal of Global Positioning System, Vol.11, No.2:127-144, 2012. [Режим доступа]: https://www.researchgate.net/publication/260790060_A_LabVIEW-Based_GPS_Receiver_Development_and_Testing_Platform_with_DSP_Peripherals_Case_study_with_C6713_DSK
[12] Albert A. Alabov, Sergey V. Nosachev, Viktor P. Zharov, Vsevolod A. Minko, Using the SimInTech dynamic modeling environment to build and check the operation of automation systems, 2018. [Режим доступа]: https://www.researchgate.net/publication/328787494_Using_the_SimInTech_dynamic_modeling_environment_to_build_and_check_the_operation_of_automation_systems.

Complex engineering and technological processes modeling system for the technological design procedures

K. D. Barseghyan, Yu.A. Andrienko

Abstract: ***The purpose of this work is to create a software development methodology for implementing the choice of solutions in the design of complex engineering and technological systems. The approaches of simulation modeling technology (IM), identification of the current state of its application in the engineering industry, comparative analysis of software (software) implementing IM technology, and the study of the most successful practical cases performed using the analyzed software are discussed. The analysis of the mathematical apparatus of the software considered at the previous stage is given, as well as the necessity of developing and forming the concept of the proposed solution, as well as setting requirements for the system under development and its functional and non-functional capabilities. The architecture of the software being developed is described, the generalized development process is presented, and the results of software testing are presented.***

Keywords – simulation modeling, production and technological process, design of production lines.

REFERENCES

[1] N.A. Polyakov, History of Simulation Modeling, pp. 1, 2012

[2] H. Bourbouh, P. L. Garoche, C. Garion, Automated analysis of Stateflow models. EPiC Series in Computing, v. 46, p. 144-161, 2017.

[3] Y.Z. Talukder, Model-based design of automatic control systems in engineering education, pp. 3, 2013.

[4] N.P. Demenkov, Model-oriented design of control systems, p. pp. 2.

[5] Internet resource “Wikipedia: V-Model.” [Access mode]: https://ru.wikipedia.org/wiki/V-Model.

[6] Internet resource “Exponenta: Simulink”. [Access mode]: https://exponenta.ru/simulink.

[7] Anylogic Internet resource. [Access mode]: https://www.anylogic.com.

[8] Internet resource “SimInTech: Reference System”. [Access mode]: https://help.simintech.ru/#o_simintech/browsers.html.

[9] Internet resource “Chempute software: ChemCAD”. [Access mode]: https://chempute.com/chemcad.html.

[10] Mohammad Vajdi, Farhad Sadegh Moghanlou, Fariborz Sharifianjazi, Mehdi Shahedi Asl, Mohammadreza Shokouhimehr, A Review on the Comsol Multyphysics studies of heat transfer in advanced ceramics, Journal of Composites and Compounds, p. 35-43, 2020. URL: https://www.researchgate.net/publication/340437687_A_review_on_the_Comsol_Multiphysics_studies_of_heat_transfer_in_advanced_ceramics.

[11] Arpine Soghoyan and David Akopyan, A LabVIEW-Based GPS Receiver Development and Testing Platform with DSP Peripherals: Case study with C6713 DSK, Journal of Global Positioning System, Vol.11, No.2:127-144, 2012. URL: https://www.researchgate.net/publication/260790060_A_LabVIEW-Based_GPS_Receiver_Development_and_Testing_Platform_with_DSP_Peripherals_Case_study_with_C6713_DSK

[12] Albert A. Alabov, Sergey V. Nosachev, Viktor P. Zharov, Vsevolod A. Minko, Using the SimInTech dynamic modeling environment to build and check the operation of automation systems, 2018. URL: https://www.researchgate.net/publication/328787494_Using_the_SimInTech_dynamic_modeling_environment_to_build_and_check_the_operation_of_automation_systems.