ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
4
2022
Том 13

Рисунки к статье
А. Н. Годунова, И. И Хоменкова, В. Г. Щепкова, А. В. Хорошилова
«КОНФИГУРИРУЕМАЯ ТЕСТОВАЯ СИСТЕМА ДЛЯ ОСРВ СЕМЕЙСТВА БАГЕТ»

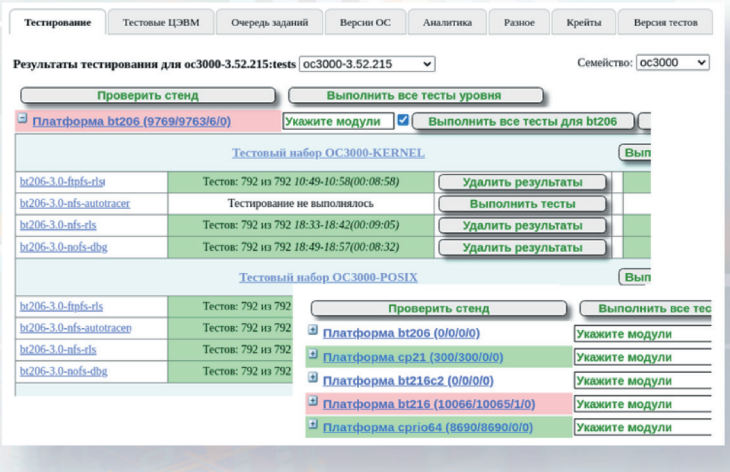

Рис. 4. Главная страница ТС в развернутом и компактном видах

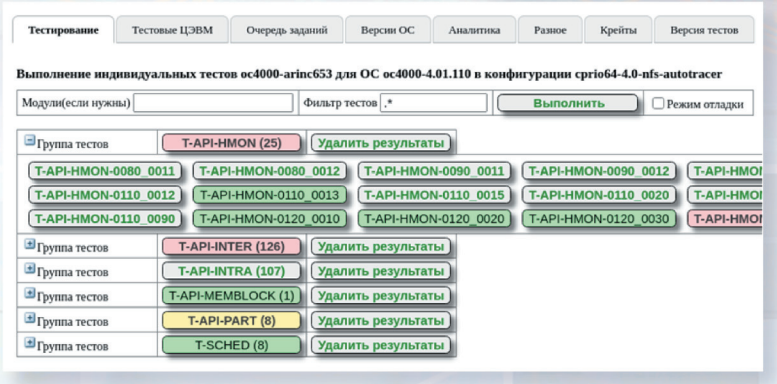

Рис. 5. Страница выбора одиночного теста или группы тестов из тестового набора

Рисунки к статье
K. I. Gaydamaka, A. D. Belonogova
«APPLYING UNSUPERVISED MACHINE LEARNING ALGORITHMS TO ENSURE REQUIREMENTS CONSISTENCY»

Fig. 2. K-means clustering

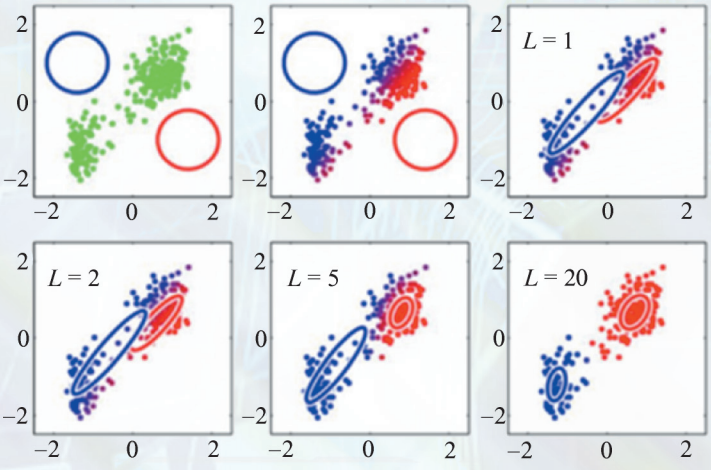

Fig. 3. EM Clustering

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в подписных агентствах (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции (для юридических лиц).
Тел.: (499) 270-16-52.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 13
N 4
2022

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.05 DOI: 10.17587/prin.13.155-167

В. И. Шелехов, канд. техн. наук, зав. лаб., vshel@iis.nsk.su,
Институт систем информатики им. А. П. Ершова СО РАН,
Новосибирский государственный университет

Автоматное программирование на базе системы моделирования и верификации Event-B

Представлен новый язык автоматного программирования, построенный расширением языка спецификаций Event-B. При разработке моделей в Event-B появляется возможность использования методов автоматного программирования в дополнение к популярному методу уточнений. Технология автоматного программирования на базе Event-B демонстрируется на примере задачи управления движением на мосту из руководства по системе Event-B. Предложено более простое решение с верификацией в инструменте Rodin. Эффективность методов верификации в Event-B подтверждена нахождением трех нетривиальных ошибок в приведенном решении.

Ключевые слова: *автоматное программирование, Event-B, уточнение, требования, дедуктивная верификация, трансформации программ, функциональное программирование*

Введение

Event-B [1] — это метод формальной спецификации и верификации систем в программной и системной инженерии, успешно используемый при разработке производственных систем управления, особенно в железнодорожном транспорте и метрополитене. Система Event-B реализована на платформе Rodin [2]. Метод верификации в Event-B гарантирует обнаружение многих серьезных ошибок. Несмотря на большое число разнообразных приложений, существует ряд факторов, сдерживающих широкое внедрение подхода Event-B, что отмечается в обзоре [3].

Автоматное программирование [4—8] ориентировано на класс реактивных систем, в частности, систем управления. Автоматная программа определяет автомат в виде гиперграфовой композиции сегментов кода. Технология автоматного программирования предлагает комплекс методов для разработки, оптимизации и верификации автоматных программ.

Построен новый язык автоматного программирования расширением языка спецификаций Event-B. Автоматная программа легко кодируется в языке спецификаций Event-B. Это дает возможность использования технологии автоматного программирования в разработке систем управления для критической инфраструктуры на платформе Rodin [2]. Данный стиль разработки иллюстрируется на примере задачи управления движением автомобилей на мосту из руководства по системе Event-B [1]. Предложено более простое решение с верификацией в инструменте Rodin. Эффективность методов верификации в Event-B подтверждена нахождением трех нетривиальных ошибок в приведенном решении.

В разд. 1 настоящей статьи определен класс программ-процессов в контексте других классов программ, описаны язык и технология автоматного программирования. В разд. 2 представлен метод Event-B. Далее в разд. 3 описано построение языка автоматного программирования на базе языка спецификаций Event-B. В разд. 4 даны два решения задачи управления движением автомобилей на мосту из руководства по Event-B [1]. Проиллюстрированы уточнения из руководства [1]. Описана разработка автоматной программы, исправляющей недостатки реализации в Event-B. В разд. 5 описана верификация автоматной программы в инструменте Rodin [2]. Обзор работ дан в разд. 6. В заключении описаны итоговые положения и замечания по технологии автоматного программирования на базе системы Event-B.

1. Автоматное программирование

1.1. Классы программ

Автоматные программы требуют других методов разработки и верификации, нежели программы вычисления некоторого результата по набору аргументов. Например, метод верификации Флойда—Хоара [9, 10] не применим для автоматных программ. Существуют разные классы программ со своими особенными методами разработки, спецификации, верификации, моделирования и оптимизации.

Классификация программ [11] определяет невзаимодействующие программы (или программы-функции), реактивные системы (или программы-процессы), языковые процессоры и операционные среды. Отметим, что такая классификация неполна и не покрывает существующего разнообразия видов программ.

Класс программ-функций (невзаимодействующих программ). Программа принадлежит этому классу,

если она не взаимодействует с внешним окружением. Точнее, если возможно перестроить программу таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод был собран в конце программы. Программа обязана всегда завершаться, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Следовательно, программа определяет функцию, вычисляющую по набору входных данных (аргументов) некоторый набор результатов.

Класс программ-процессов (реактивных систем). Программа-процесс является реактивной системой, реагирующей на определенный набор событий (сообщений) во внешнем окружении программы. Программа-процесс является либо автоматной программой, либо она определяется в виде композиции нескольких автоматных программ, исполняемых параллельно и взаимодействующих между собой через прием/посылку сообщений и разделяемые переменные.

Автоматная программа состоит из одного или нескольких сегментов. *Сегмент* имеет один вход, помеченный меткой — *управляющим состоянием*. Сегмент имеет один или несколько выходов. Автоматная программа определяет автомат в виде гиперграфа с набором управляющих состояний в качестве вершин и набором сегментов в качестве ориентированных гипердуг. *Состояние* автоматной программы определяется значениями набора переменных, модифицируемых в программе.

Языковые процессоры — это интерпретаторы программ, компиляторы, оптимизаторы, трансформаторы и другие процессоры.

Класс программ "операционная среда". Программа данного класса является автоматом общего вида. Это, например, операционная система в целом или ее фрагмент, например, система файлов. Операционная среда в каждый текущий момент функционирования имеет несколько активных позиций. Каждая позиция функционирует как независимая автоматная программа со своим набором управляющих и внутренних состояний.

Полезность универсальных методов. Метод признается универсальным, если он применим для некоторого класса программ. Метод полезен для конкретной программы, если его применение дает преимущества по сравнению с некоторой типовой технологией, не использующей данный метод.

Автоматное программирование универсально. Любая программа-функция может быть запрограммирована в виде автоматной программы, которая при этом будет значительно сложнее аналогичной функциональной или императивной программы. Автоматное программирование не полезно для программ-функций.

Полезность метода объектно-ориентированного программирования определяется возможностью инкапсуляции (упрятывания) разнообразных деталей реализации, предоставляя наружу более простой интерфейс. Если не происходит существенного упрощения внутреннего интерфейса программы, то объектно-ориентированные конструкции просто загромождают программу, удлиняя и усложняя ее. Между тем имеется достаточно стойкая тенденция, иногда подкрепленная местными стандартами в разных средах разработчиков и провоцируемая такими языками, как Java, злоупотребления объектно-ориентированным программированием. Кроме того, в редких случаях объектно-ориентированное программирование требуется в полном объеме. В подавляющем большинстве случаев можно было бы ограничиться более простыми возможностями на уровне абстрактных типов данных [12].

1.2. Язык автоматного программирования

Программа-процесс содержит фрагменты, соответствующие программам-функциям. Поэтому язык автоматного программирования должен быть построен как расширение некоторого *базисного* языка (императивного или функционального) для класса программ-функций. Таким способом определено автоматное расширение [5, 6] языка предикатного программирования P [4, Разд. 10]. Язык автоматного программирования можно построить расширением любого языка для класса программ-функций и даже расширением языка спецификаций.

Программа-процесс состоит из секций и автоматных программ. *Секция* определяется следующей конструкцией:

section <Имя секции> **extends** <Имя секции>
{ <Описания типов, констант, переменных
и инвариантов> }

В секции описываются переменные состояния программы (возможно, части состояния), а также константы и типы. Обычно секция помещается перед автоматной программой. Имя секции может отсутствовать. Секция может быть построена расширением другой секции при наличии **extends** <Имя секции>. В секции также помещаются общие инварианты для всей программы.

Автоматная программа определяется следующей конструкцией:

process <Имя программы>(< Описания аргументов>)
{ <Сегменты кода> }

Аргументы могут отсутствовать. Автоматная программа может быть вызвана внутри другой автоматной программы. Произвольный <Сегмент кода> представляется следующей конструкцией:

<Имя управляющего состояния>:
inv <Формула>; <Оператор>

Исполнение <Оператора> завершается оператором перехода вида #M, реализующим переход на начало сегмента с управляющим состоянием M. <Формула> определяет инвариант, который должен быть истинным в начале сегмента для данного управляющего состояния.

Представленная структура управления автоматной программы аналогична используемой в языке Фортран, применяемом в основном для задач вы-

числительной математики, но не для реактивных систем.

Сегмент, код которого не содержит вызовов других процессов, является *атомарным*: исполнение сегмента должно быть единым, неделимым актом; это означает, что до завершения исполнения сегмента все другие параллельно исполняемые процессы останавливаются.

Сегмент кода следующего вида:

M: if (<условие$_1$>&...& <условие$_n$>)
{ <действие$_1$>; ...; <действие$_m$> **#L** }

может быть записан в виде правила [7]:

M: <условие$_1$>,..., <условие$_n$> →
<действие$_1$>, ..., <действие$_m$> **#L**

Сегмент вида **M: if** (C) A **else** B **#L** может быть представлен парой правил:

M: C → A **#L**
M: B **#L**

Правила исполняются в порядке их следования. Второе правило **M:** B **#L** может сработать только при ложном условии C.

Для операторов A и B оператор A || B определяет параллельное исполнение операторов A и B. Оператор A | B определяет недетерминированный выбор для исполнения одного из операторов, A или B.

В языке автоматного программирования определяются также операторы приема и посылки сообщений, действия со временем: установка таймера, оператор задержки по времени и др.

1.3. Спецификация и верификация автоматных программ

Методы верификации, успешно применяемые для программ-функций, в частности, логика Хоара—Флойда [9, 10], непригодны для верификации автоматных программ. Эти методы могут применяться лишь для фрагментов автоматных программ, соответствующих программам-функциям.

Задача разработки программы-процесса формулируется в виде набора требований. *Требование* — утверждение, определяющее потребность и связанные с ней измеримые условия и ограничения. Требования к реактивной системе включают требования окружения и функциональные требования, определяющие поведение системы. Существенными являются также нефункциональные требования надежности, безопасности, защищенности, отсутствия дедлоков и др. Разработка требований должна проводиться в соответствии со стандартом ISO/IEC/ IEEE 29148 [13].

Спецификация программы-процесса является частью требований или непосредственно вытекает из требований. Спецификация определяется общими инвариантами и инвариантами управляющих состояний. Отметим, что инварианты управляющих состояний и общие инварианты принципиально отличаются от инвариантов циклов императивной программы. Инварианты управляющих состояний часто являются слабыми или отсутствуют, т. е. оказываются тождественно истинными.

Далее будет показано, каким образом инвариант управляющего состояния преобразуется в общий инвариант.

Общий инвариант формулируется для автоматной программы в секции непосредственно перед автоматной программой. Общий инвариант должен быть истинным в начале каждого сегмента. Общий инвариант может быть представлен темпоральной формулой.

Истинность общего инварианта гарантируется доказательством следующей серии формул корректности. Для каждого управляющего состояния необходимо доказать, что из истинности общего инварианта в начале соответствующего сегмента следует истинность общего инварианта для модифицированных значений переменных на каждом выходе из данного сегмента.

Для любого фрагмента программы, соответствующего программе-функции, необходимо доказать истинность предусловия для данного фрагмента. Например, для операции деления необходимо доказать, что выражение, являющееся делителем, не равно нулю.

Другие свойства программ-процессов, которые подлежат верификации, — это отсутствие взаимной блокировки процессов (дедлоков) и завершение конечных процессов.

Доказательство истинности всех перечисленных видов формул корректности позволяет избежать многих серьезных ошибок, но оно не гарантирует полной корректности программы в отличие от верификации программ-функций, где правильность спецификации и доказательство формул корректности гарантируют корректность программы.

1.4. Технология автоматного программирования

Разработка программы-процесса начинается с определения набора требований. Фиксируются объекты и их атрибуты. Формулируются требования окружения, функциональные требования, требования безопасности и др. Достаточно часто автоматная программа строится простым переписыванием функциональных требований. Их удобнее записывать в виде логических правил [7]. Требования безопасности формализуются в виде общих инвариантов.

Основной метод автоматного программирования — это метод декомпозиции автоматной программы в виде нескольких сегментов кода. Каждый сегмент кода автоматной программы соответствует некоторому управляющему состоянию и реализует независимую более простую подзадачу. Сужения для подзадачи фиксируются инвариантом управляющего состояния.

Гиперграфовая композиция автоматной программы является более общей по сравнению с композициями операторов в традиционных языках, таких как Си. Гиперграфовая композиция является предельно гибкой. Любую автоматную программу можно предста-

вить композицией двух сегментов. Гиперграфовая композиция трех независимых процессов позволила упростить программу управления лифтом [6].

Сегмент кода, реализующий программу-функцию и имеющий несколько выходов, является *гиперфункцией* [4, 14, 15]. Гиперфункции обеспечивают дополнительные возможности оптимизации программ.

Для получения более эффективной программы применяются эквивалентные автоматные трансформации, существенно меняющие структуру автоматной программы [8].

Для упрощения интерфейса программы полезно использовать методы объектно-ориентированного и аспектно-ориентированного программирования. Отметим, что редко требуется использовать объектно-ориентированное программирование в полном объеме — достаточно ограничиться типовыми конструкциями на уровне абстрактных типов данных [12].

Стиль языка Фортран для автоматной программы в целом контрастирует со стилем языка Си внутри сегмента кода, особенно при использовании циклов вида **while** и **for**. Желательно ограничить использование циклов **while** и **for** в автоматных программах.

2. Система Event-B и платформа Rodin

Event-B [1] — это метод формальной спецификации и верификации систем в программной и системной инженерии с использованием нотации теории множеств и логики первого порядка. Спецификация на языке Event-B состоит из компонентов двух видов: контекстов и машин. *Контекст* определяет множества и константы — статическую часть спецификации. *Машина* содержит: переменные, инварианты, события. События машины определяют процесс в виде недетерминированного автомата. Значения переменных формируют текущее *состояние* процесса.

Текущее состояние спецификации может быть изменено событием. *Событие* определяет охранные условия и действия. *Охранные условия* ограничивают множество возможных состояний, в которых данное событие может произойти. *Действия* реализуют присваивания переменным. Эффект более сложного действия можно определить before-after-предикатом, связывающим значения исходных и модифицированных переменных. Для всякого события инварианты должны оставаться истинными после модификаций переменных действиями события.

Все события атомарны и могут реализоваться, если истинны их охранные условия. Если одновременно истинны охранные условия нескольких событий, то только одно из них может исполниться в данный момент; событие для исполнения выбирается недетерминированным образом.

Спецификация Event-B состоит из последовательности машин, в которой очередная машина является *уточнением* предыдущей машины. Очередная машина содержит новые переменные и события, расширяющие предыдущую машину. Причем поведение новой машины в проекции на наследуемые объекты и события полностью идентично поведению предыдущей машины.

Платформа Rodin [2] определяет среду для разработки и верификации спецификаций на языке Event-B. Доказательство генерируемых по спецификации формул корректности поддерживается автоматическими и интерактивными средствами доказательства с применением SMT-решателей [2]. Степень и возможности автоматизации доказательства существенно выше, чем в системах доказательства PVS [16], Why3 [17] и Coq [18]. В частности, автоматизировано применение эквивалентных замен. В дереве текущего процесса доказательства явно обозначены позиции, по которым пользователь может продолжить доказательство, выбрав одну из предлагаемых альтернатив.

В Event-B имеются средства обнаружения ситуаций взаимной блокировки (дедлока), а также доказательства завершения конечных процессов.

3. Язык автоматного программирования на базе Event-B

В автоматном программировании можно представить любую технологию для программ-процессов. Далее определим построение нового языка автоматного программирования расширением языка спецификаций Event-B.

Произвольное событие далее будем представлять в виде оператора:

if (<Охранные условия>) { <Действия> }

или в виде правила:

<Охранные условия> → <Действия>

В типичном случае действия реализуют присваивание нескольким переменным. Например, { c := C; d := D }. Выражения C и D вычисляются и их значения одновременно присваиваются переменным c и d.

Допустим, машина M состоит из событий A, B, E и F. Тогда машину M будем представлять следующим процессом с единственным управляющим состоянием cycle:

```
process M
{ cycle: [nA:] A | [nB:] B | [nE:] E | [nF:] F #cycle }
```

Здесь nA, nB, nE и nF — имена событий A, B, E и F. Имена событий, альтернатив недетерминированной композиции, представлены здесь в квадратных скобках в отличие от имен управляющих состояний.

В новом языке автоматного программирования будем использовать примитивные типы BOOL, INT, NAT языка Event-B и оператор присваивания <Переменная> := <Выражение>.

Некоторые конструкции языка Event-B заменяются другими, более привычными. Предикат x ∈ BOOL будем записывать в виде x: BOOL. Не используется событие INITIALISATION. Начальная инициализация переменных реализуется в секции перед автоматной

программой. Например, x: BOOL := false. Вместо конструкции partition со сложной семантикой используется описание типа перечисления **enum** в стиле языка Си.

4. Управление движением автомобилей по мосту

Задача управления движением автомобилей по мосту представлена в качестве первого примера в известной книге J.-R. Abrial, являющейся руководством по системе Event-B [1]. На базе этого примера определяются основные положения технологии Event-B.

4.1. Требования к системе управления

Содержательное описание. Имеется мост, соединяющий материк и остров. Мост узкий и позволяет двигаться автомобилям по нему только в одну сторону. Имеются два светофора, установленных при въезде на мост с материка и с острова. У каждого светофора два цвета: красный и зеленый. Автомобилям запрещено движение на красный светофор при въезде на мост. Имеются четыре сенсора. Каждый сенсор находится на некотором участке автомобильной трассы и способен фиксировать ситуацию, когда автомобиль находится на этом участке трассы. Первый сенсор находится перед въездом на мост с материка. Второй сенсор — на мосту перед съездом на остров. Третий сенсор — перед въездом на мост с острова. Четвертый сенсор на мосту перед съездом на материк.

Число автомобилей, которые могут находиться на острове, ограничено. Необходимо построить контроллер, который переключает светофоры, используя показания сенсоров, и таким образом обеспечивает безопасное движение автомобилей по мосту.

Далее проведем анализ, детализацию и именование требований.

Объекты окружения: материк, остров, мост, 2 светофора, 4 сенсора.

Атрибуты светофора: красный; зеленый.

Атрибуты сенсора: on — сенсор включен; off — сенсор выключен.

Имена светофоров: mtl — светофор на материке при въезде на мост; itl — светофор на острове при въезде на мост.

Имена сенсоров: mlOut — сенсор при въезде на мост с материка; ilIn — сенсор при съезде с моста на остров; ilOut — сенсор при въезде на мост с острова; mlIn — сенсор при съезде с моста на материк.

Определение. Сенсор *отключается*, если он изменяет свое значение с on на off.

Каждый сенсор сопряжен с некоторым участком автомобильной трассы. Сенсор включен, если на данном участке находится автомобиль. Отключение сенсора означает, что автомобиль съехал с данного участка.

Сформулируем функциональные требования.

RF1. Отключение сенсора mlOut — автомобиль въехал на мост с материка.

RF2. Отключение сенсора ilIn — автомобиль съехал с моста на остров.

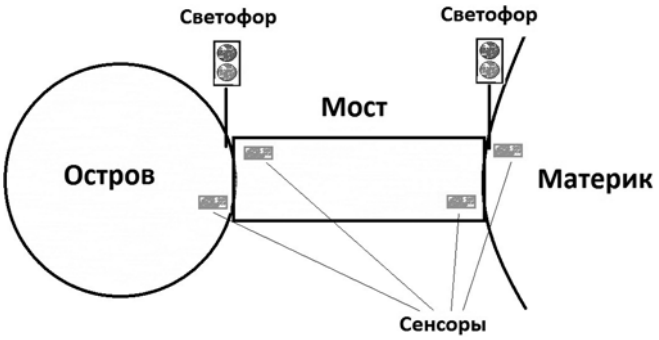

Рис. 1. Схема моста со светофорами и сенсорами

RF3. Отключение сенсора ilOut — автомобиль въехал на мост с острова.

RF4. Отключение сенсора mlIn — автомобиль съехал с моста на материк.

RF5. Число автомобилей на острове ограничено.

RF6. Движение автомобилей по мосту возможно либо с материка на остров, либо с острова на материк.

Определим требования безопасности.

RS1. При красном светофоре mtl запрещено движение с материка на мост.

RS2. При красном светофоре itl запрещено движение с острова на мост.

Представленная система управления движением по мосту в целом и архитектура оборудования (рис. 1) в частности имеют серьезные недостатки и не могут использоваться на практике. Ничто не может помешать автомобилю встать на сенсор, а затем поехать в обратном направлении. Автомобиль может также заехать на мост, а потом съехать с него задним ходом. Поэтому данную задачу построения контроллера управления движением по мосту можно рассматривать только как учебную.

4.2. Разработка от простейшей модели

Построение спецификации управления движением по мосту представлено в книге [1] в виде простейшей модели и последующих трех уточнений. Для иллюстрации технологии Event-B воспроизведем начальную модель и первые два уточнения на языке автоматного программирования.

В начальной модели мост и остров рассматриваются как единое целое. В модели фиксируются автомобили, уходящие с материка, и автомобили, приходящие на материк. Пусть n — число автомобилей на мосту и на острове. Введем константу d, определяющую максимальное число автомобилей на острове. Представим (рис. 2) автоматную программу для начальной машины с двумя событиями: уход автомобиля с материка и приход автомобиля на материк.

Легко обнаружить, что в модели есть ошибки. Первая ошибка: нет гарантии, что после действия n:=n+1 не будет превышен лимит автомобилей d. Эффективность метода Event-B [1] в том, что он позволяет обнаруживать все ошибки такого рода в процессе доказательства формул корректности. В данном

```
section St0 {
    const d: NAT
    n: NAT
    inv  n ≤ d
}
process carBridge0er {
   cycle: [ML_out:] n := n+1 |
          [ML_in:] n := n-1
          #cycle
}
```

Рис. 2. Начальная модель с ошибками

```
section St0 {
    const d : NAT  // максимальное число автомобилей на острове
    axiom d > 0
    n: NAT := 0
    inv  n ≤ d
}
process carBridge0 {
   cycle: [ML_out:] n < d → n := n+1 |
          [ML_in:]  n > 0 → n := n-1
          #cycle
}
```

Рис. 3. Исправленная начальная модель

```
section St1 extends St0{
    a: NAT := 0
    b: NAT := 0
    c: NAT := 0
    inv a+b+c = n
    inv a=0 ∨ c = 0
}
process carBridge1 refines carBridge0 {
   cycle: [ML_out:]  a+b<d,  c=0 → a:=a+1 |
          [ML_in:]   c>0  →  c:=c-1 |
          [IL_out:]  b>0, a=0 →  b:=b-1, c:=c+1 |
          [IL_in:]   a>0  →  a:=a-1, b:=b+1
          #cycle
}
```

Рис. 4. Первое уточнение

случае будет представлена формула корректности: n+1⩽d, которая здесь недоказуема. Для исправления ошибки необходимо добавить условие: n<d. Вторая ошибка: для действия n:=n–1 будет сгенерирована формула корректности n-1 ∈ NAT, которая оказывается недоказуемой. Необходимо вставить условие: n>0. Третья ошибка: в начальный момент работы машины не выполняется формула корректности: n⩽d. Причина в том, что переменная n не инициализирована. В секции St0 необходимо вставить инициализацию: n:= 0. После исправления данных ошибок появляется четвертая ошибка. При поиске блокировок (дедлоков) генерируется формула корректности: n⩽d ⇒ n<d ∨ n>0, которая недоказуема при d=0. В случае d=0 действительно возникает дедлок, поэтому необходимо условие d>0 в виде аксиомы. На рис. 3 дана исправленная начальная модель.

Определим уточнение начальной модели (рис. 4). Будем различать автомобили, находящиеся на мосту и на острове. Заменим старую переменную n тремя новыми переменными: a — число автомобилей, движущихся по мосту с материка на остров; b — число автомобилей на острове; c — число автомобилей, движущихся по мосту с острова на материк. Вводится *связующий инвариант* a+b+c=n, определяющий отношение между старыми и новыми переменными. Появляются два новых события: автомобиль съезжает с моста на остров (IL_in) и автомобиль въезжает на мост с острова (IL_out).

Можно видеть, что события ML_out и ML_in в уточненной модели отличаются от этих же событий в начальной модели. Для этих событий в Event-B генерируются формулы корректности, доказательство которых гарантирует идентичность поведения старых и новых событий. Корректность уточнения для добавленных событий IL_out и IL_in будет обеспечена, если эти события не меняют значений старых переменных, т. е. значения переменной n.

Следующая модель (рис. 5) строится как уточнение модели carBridge1. Это уточнение другого вида. В модели carBridge1 уточнялись структуры данных. Здесь же происходит добавление новых структур данных и новых событий.

Появляются светофоры: mtl — на материке при въезде на мост; itl — на острове при въезде на мост. Инвариант mtl=red ∨ itl=red соответствует требованию RF6. Два других инварианта отражают следующие правило корректности уточнения: охранное условие некоторого события в уточненной модели должно быть не слабее аналогичного условия в предыдущей модели.

На пустом мосту новые события Mtl_green и Itl_green переключают светофоры так, чтобы стало возможным движение в противоположную сторону. Далее, поскольку мост все еще пустой, может сработать другое событие, переключающее светофоры. В результате получим бесконечный процесс быстрого

```
section St2 extends St1{
    type Colour = enum {red, green}
    mtl: Colour := red
    itl: Colour := red
    inv mtl=red ∨ itl=red
    inv mtl=green => a+b<d & c=0
    inv itl=green => b>0 & a=0
}
process carBridge2 refines carBridge1{
   cycle: [ML_out:] mtl=green → a:= a+1 |
          [ML_in:]  c>0  → c:=c-1 |
          [IL_out:] itl=green  →  b:=b-1, c:= c+1 |
          [IL_in:]  a>0 →  a:=a-1, b:=b+1 |
          [Mtl_green:]  mtl=red, c=0, b<d  →  mtl:=green, itl:=red |
          [Itl_green:]  itl=red, a=0, b>0 →  itl:=green, mtl:=red
          #cycle
}
```

Рис. 5. Второе уточнение

переключения светофоров. Чтобы не допустить такую ситуацию, дальнейшее переключение светофоров блокируется в модификации машины carBridge2 до появления автомобиля на мосту с противоположной стороны. Данное решение неудовлетворительно, поскольку ожидание автомобиля с противоположной стороны может оказаться долгим.

Правильное решение следующее. Изменение направления движения возможно, если с противоположной стороны имеется автомобиль, стоящий на сенсоре в ожидании зеленого светофора.

В третьем уточнении добавляются сенсоры. Модель оказалась сложной. Для проведения уточнения потребовалось ввести 11 новых переменных.

Уточнение в Event-B фактически единственный метод. Он позиционируется как универсальный и эффективный для декомпозиции сложных моделей. В данном примере из руководства [1] уточнения были введены в учебных целях. Третье уточнение оказалось неоправданно сложным. Было бы намного проще построить модель, не прибегая к уточнениям.

Метод уточнений является эффективным для иерархических моделей. Построение искусственной иерархии обычно не дает хорошего результата. Метод уточнений не всегда полезен.

4.3. Разработка по технологии автоматного программирования

Изменения числа автомобилей на мосту и на острове определяются через изменения показаний сенсоров. Функционирование сенсоров естественно определить независимыми процессами, вычисляющими число автомобилей на мосту и на острове. Управление светофорами удобнее реализовать другим процессом. Полная программа-процесс управления движением автомобилей на мосту представляется параллельной композицией пяти автоматных программ:

```
process carBridge {
    LightControl || ML_out || IL_in || IL_out || ML_in
}
```

Рассмотрим автоматную программу управления светофорами LightControl. Очевидными являются следующие три управляющих состояния:

- zero — мост пуст, оба светофора красные;
- right — реализуется движение слева направо, т. е. с острова на материк;
- left — реализуется движение справа налево, т. е. с материка на остров.

Каждое управляющее состояние определяет независимую подзадачу, более простую по сравнению с исходной задачей.

В состоянии right реализуется управление движением с острова на материк. Эта задача намного проще исходной задачи. Например, нет необходимости проверять ограничение числа автомобилей на острове. Если мост становится пустым, необходимо перейти в состояние zero.

В состоянии left реализуется управление движением с материка на остров. Если мост становится пустым, происходит переход в состояние zero. При появлении автомобиля, въезжающего на мост с материка, также проверяется ограничение a+b⩽d. В ситуации a+b=d светофор mtl устанавливается красным. Для того чтобы перейти в состояние zero, необходимо сначала освободить мост. Освобождение моста контролируется в дополнительном управляющем состоянии leftscan.

В состоянии zero мост пуст. Оба светофора красные. Переход в состояние right или left становится возможным, когда у моста на сенсоре останавливается автомобиль в ожидании зеленого светофора. Чтобы отслеживать остановку автомобиля на сенсоре перед мостом, вводятся две новые переменные. Переменная ml_out=true, если сенсор mlOut включен, т. е. на нем со стороны материка в данный момент находится автомобиль. Переменная il_out = true, если сенсор ilOut включен — на нем находится автомобиль со стороны острова.

Очевидно, что после перехода с zero на right или left немедленно произойдет возврат на zero, потому что автомобиль не успеет заехать на пустой мост. Получим быстрое мерцание светофора с зеленого на красный. Чтобы избежать мерцания светофоров, переход с zero реализуется в новые состояния right0 и left0, где ожидается появление на мосту первого автомобиля.

Если b=d в состоянии left0, то при появлении первого автомобиля на мосту будет нарушено ограничение a+b⩽d. В связи с этим, при переходе с zero на left0 необходимо дополнительно проверять условие b<d.

Возникает ситуация блокировки (дедлока) в состояниях right0 и left0, если первый автомобиль появился на мосту и успел выехать с моста раньше, чем сработал сегмент кода в состояниях right0 или left0. Подобное возможно, когда процесс LightControl будет приостановлен. Во избежание блокировки введен признак пустого моста empty, который гасится в состояниях right0 или left0. При истинном значении empty автомобиль не может покинуть мост.

На рис. 6 представлена секция для полной программы управления движением по мосту. Константа d,

```
section St0 {
    const d : NAT
    axiom d > 0
    a: NAT := 0
    b: NAT := 0
    type Colour = enum {red, green}
    mtl: Colour := red
    itl:  Colour := red
    ml_out: BOOL := false
    il_out: BOOL := false
    empty: BOOL := false
    inv  a + b ≤ d
    inv mtl = red ∨ itl = red
}
```

Рис. 6. Секция для полной программы

```
process LightControl {
    zero: inv a=0 & mtl = red & itl = red & not empty;
          if (il_out) {itl := green; empty := true #right0}
          else if (ml_out & b< d) {mtl := green; empty := true #left0}
          else #zero
    right0: inv mtl = red & itl = green & empty;
           if (a=0) #right0 else { empty := false #right}
    right:  inv mtl = red & itl = green & not empty;
            a=0 → itl := red #zero
    left0:  inv itl = red & b< d & empty;
           if (a=0) #left0 else { empty := false #left }
    left:   inv itl = red & not empty;
           if (a+b=d) {mtl := red; #leftscan}
           else if (a=0) {mtl := red #zero }
           else #left
    leftscan: inv mtl = red & itl = red & not empty;
             a=0 → #zero
}
```

Рис. 7. Автоматная программа LightControl

переменные b, mtl, и itl определяются также как и в описанной выше реализации. Переменная a определяет число автомобилей на мосту в любом из направлений движения. Переменная c исключена.

На рис. 7 представлена автоматная программа LightControl. Каждое управляющее состояние снабжается инвариантом, который должен быть истинным в начале соответствующего сегмента.

4.4. Программы управления сенсорами

Простейшее функционирование сенсора представляется программой из двух сегментов включения и выключения сенсора:

```
off: #on
on: #off
```

В действительности, автоматные программы управления сенсорами нагружены вычислением значений переменных a, b, ml_out и il_out. Программы ML_out и IL_out реализуют переключения с on на off при зеленом светофоре.

Программы управления сенсорами должны успевать отслеживать изменения показаний сенсоров. Это следует сформулировать в виде следующих требований.

RF7. Включение сенсора при появлении на нем автомобиля должно быть зафиксировано раньше, чем автомобиль съедет с сенсора.

RF8. Выключение сенсора при съезде с него автомобиля должно быть зафиксировано раньше, чем другой автомобиль заедет на этот сенсор.

На пустом мосту, при a=0, невозможна ситуация "автомобиль покидает мост", т. е. переход с on на off. В действительности такого и не произойдет. Если автомобиль находится на сенсоре или заезжает на него, то мост не пуст. Поэтому необходимо вставить дополнительное условие: a>0. Без него оператор a:=a–1 окажется некорректным. Аналогичным образом, необходимо вставить условие b>0 для блокировки ситуации въезда на мост с острова при отсутствии на нем автомобилей.

Программы управления сенсорами представлены далее.

```
process ML_out {
    off: ml_out: = true #on
    on: if (mtl = green){ a:=a+1; ml_out:=false #on1
        } else #on
    on1: if (a + b = d){ mtl:=red #off } else #off
}
```

В программе ML_out при зеленом светофоре mtl в состоянии on автомобиль въезжает на мост, увеличивая число автомобилей на мосту. Если их число на мосту и острове достигает d, то тут же в состоянии on1 светофор mtl переключается на красный. В принципе, такое переключение должно немедленно произойти в состоянии left программы LightControl. Если удалить установку mtl:=red в состоянии on1, то потенциально, хотя и крайне маловероятно, следующий автомобиль успеет съехать с материка на мост ранее, чем программа LightControl выполнит оператор mtl:=red в состоянии left.

```
process IL_in {
    off: a>0, not empty, itl=red → #on
    on: a:=a-1; b:=b + 1 #off
}
```

В программе IL_in в сегменте off используется условие **not** empty для предотвращения дедлока в состоянии left0. Отметим, что в случае истинного значения empty автомобиль все равно съедет с моста, однако исполнение сегмента off будет задержано до погашения признака empty в сегменте left0. Такая задержка при отсутствии других автомобилей не является критичной.

Если удалить условие itl=red в сегменте off программы IL_in, то автомобиль, только что въехавший на мост с острова, сможет вернуться назад по данному сегменту off. Понятно, что в действительности такого не произойдет, поскольку обратное движение запрещено. Здесь же приходится блокировать такую возможность вставкой условия itl=red.

```
process ML_in {
    off: itl=green, a>0, not empty → #on
    on: a:=a-1 #off
}
```

В программе ML_in используется условие itl=green. Это условие не проверяется при съезде с моста на материк в силу отсутствия там светофора. Данное условие блокирует возможность возвращения автомобиля, только что въехавшего на мост с материка.

```
process IL_out {
    off: b>0 → il_out:=true #on
    on: itl=green → b:=b-1, a:=a+1, il_out:=false #off
}
```

Когда один автомобиль заезжает на мост, а другой автомобиль съезжает с моста, две разные программы, исполняемые параллельно, будут пытаться, возможно, одновременно, изменить значение переменной a. Подобное становится критичным, когда программы

управления сенсорами реализуются разными процессорами.

Чтобы исключить возможность одновременного доступа к переменной a разными процессами, в работе [19] используется монитор для пересчета переменных a и b, управляемый семафорами.

5. Опыт верификации в системе Rodin

Программа управления движением автомобилей по мосту, представленная выше, была закодирована в виде спецификации на языке Event-B [1] и верифицирована в инструменте Rodin [2]. Доступна реализация в Rodin: https://persons.iis.nsk.su/files/persons/pages/bridge.zip

Кодирование переходов по управляющим состояниям проводится методом, аналогичным Switch-технологии [20]. Управляющие состояния программы LightControl кодируются как значения переменной state типа перечисления State:

type State = **enum** { zero, right0, right, left0, left, leftscan }

Сегмент кода **M:** A **#L** заменяется оператором **if** (state = **M**) {A; state := **L**}. Гиперграфовая композиция сегментов кода заменяется одним бесконечным циклом, тело которого — недетерминированная композиция модифицированных сегментов. Предварительно сегменты кода будет удобнее представить в виде правил. Программа LightControl преобразуется к виду, показанному на рис. 8.

Каждая альтернатива недетерминированной композиции непосредственно переписывается в виде события на языке Event-B [1]. Например, альтернатива для управляющего состояния leftscan представляется следующим событием:

```
EVENT Leftscan
WHERE
    grd1: state = leftscan
    grd2: a = 0
THEN
    act1: state := zero
END
```

Инварианты управляющих состояний могут быть преобразованы в общие инварианты. Они оказываются полезными при доказательстве формул корректности. Например, инвариант для управляющего состояния zero может быть представлен общим инвариантом:

state = zero ⇒ a=0 & mtl=red & itl=red & **not** empty

Кодирование автоматных программ для сенсоров можно провести без управляющих состояний off и on, используя переменные ml_out и il_out. Для программ IL_in и ML_in применяется автоматная трансформация [8], заменяющая композицию

```
off: A #on
on: B #off
```

одним сегментом off: A; B #of. Далее оператор A; B кодируется событием Event-B.

Пять автоматных программ, исполняемых параллельно в полной программе управления движением на мосту, кодируются в одной машине на языке Event-B. Итоговая смесь всех событий в рамках одной машины фактически реализует полное перемешивание (интерливинг) атомарных сегментов пяти автоматных программ.

Сто сгенерированных формул корректности инвариантов были доказаны автоматически в системе Rodin. Кроме одной формулы, доказанной с помощью интерактивных средств доказательства. Инварианты управляющих состояний были преобразованы в общие инварианты и добавлены к двум основным инвариантам программы.

Дальнейшая верификация проводилась применением *аниматора* — компонента Rodin, реализующего пошаговое исполнение спецификации Event-B с визуализацией текущего состояния, возможностью выбора любого из возможных вариантов исполнения и возможностью вернуться назад в процессе исполнения. Теоретически число вариантов исчисляется астрономическим числом. Однако фактически в каждый момент доступны для исполнения не более трех событий — остальные события заблокированы, поскольку их охранные условия ложные. Поэтому оказалось возможным эффективно перебрать с помощью аниматора ключевые варианты исполнения и обнаружить следующие ошибки в программе управления движением на мосту.

Сначала (первая ошибка) был обнаружен дедлок в состояниях right0 и left0, когда первый автомобиль на мосту выезжает с него раньше срабатывания сегмента в состояниях right0 или left0. Был введен признак empty для пустого моста. Вторая ошибка:

```
process LightControl {
  State state := zero;
  Cycle: [Zero1:] state=zero, il_out → itl := green, empty := true, state:=right0 |
         [Zero2:] state=zero, ml_out, b<d → mtl := green, empty := true, state:=left0 |
         [Right0:] state= right0, a ≠ 0 → empty := false, state := right |
.....................................................................................
         [Leftscan:] state = leftscan, a = 0 → state := zero
       #Cycle
}
```

Рис. 8. Кодирование управляющих состояний программы LightControl

дедлок в состоянии left0 при b=d. Для исправления ошибки введено дополнительное охранное условие b<d в состоянии zero. Третья ошибка обнаружена на одном из вариантов анимации. После въезда на мост первого автомобиля с острова этот автомобиль неожиданно возвращался на остров. Здесь оказалось возможным срабатывание сегмента off процесса IL_in. Для исправления ошибки введено дополнительное охранное условие itl=red.

Таким образом, система Rodin показала свою эффективность при верификации автоматных программ.

Четвертая ошибка обнаружена при написании статьи. В программе ML_out в сегменте on использовалось условие a+b<d для пропуска автомобиля на мост с материка. Отметим, что сенсор реагирует только на появление на нем или съезде с него автомобиля. А автомобиль может съехать только при зеленом светофоре; условие a+b<d не может воздействовать на автомобиль. После пересчета a:=a+1 в сегменте on далее сегмент left программы LightControl при истинности условия a+b=d устанавливает красный светофор. Потенциально, хотя и маловероятно, следующий автомобиль может успеть въехать на мост раньше (при зеленом светофоре), чем сегмент left выполнит оператор mtl:=red. По этой причине проверка a+b=d и установка mtl:=red должны проводиться в программе ML_out сразу же после пересчета a:=a+1 в сегменте on. После исправления данной ошибки для доказательства формул корректности потребовался дополнительный инвариант: @thm_ml_out a+b = d ⇒ mtl=red.

Удаление переменной c оказалось неправильным решением, усложняющим программу. Это решение спровоцировало третью ошибку, усложнило доказательство, а также потребовало вставить дополнительную проверку itl=green в сегменте on программы ML_in.

6. Обзор работ

Автоматное программирование. Автоматные методы программирования заложены во многих известных языках, таких как SDL [21], UML, TLA+ [22], Event-B [1], Дракон [23], а также LD, FBD и SFC, определяемых стандартом IEC 61131-3 для программируемых логических контроллеров. Большинство этих языков являются графическими.

Автоматное программирование [5—8, 20, 24] развивается исключительно в России, начиная с 1990-х годов. Оно доказало свою эффективность на множестве разнообразных приложений. Автоматная программа конструируется в виде гиперграфовой композиции сегментов кода, более общей в сравнении с композициями операторов в традиционном программировании. Каждый сегмент кода метится управляющим состоянием и реализует независимую подзадачу, условие которой частично отражается инвариантом управляющего состояния. Концепция автоматного программирования на базе управляющих состояний ранее предложена в работах [20, 24].

Задача управления движением автомобилей по мосту. Задача используется в работе [25] для иллюстрации метода построения спецификации Event-B снизу, начиная с более простых компонентов (сенсоров и светофоров), которые затем включаются в основную спецификацию. Механизм построения снизу интегрирован с техникой уточнений, реализуемой сверху. Ранее предлагались другие методы конструирования спецификации из независимых модулей. Модульная декомпозиция реализована в классическом методе B [26], однако была утеряна при разработке Event-B.

В работе [25] построены универсальные модели сенсоров и светофоров, каждая применением трех уточнений. Основная модель строится в виде цепочки из восьми машин, полученных уточнениями и включениями подмоделей. Итоговая реализация получилась громоздкой — одна из машин с включенными подмоделями длиною в 379 строк. Единственная машина, полученная из пяти автоматных программ, содержит 165 строк.

Применение уточнений при разработке модели управления движением на мосту в руководстве по Event-B [1] и в работе [25] не упрощает модель, а усложняет ее. Это обнаруживается при сопоставлении с реализацией в автоматном программировании.

Кодирование автоматных программ. Существуют разные способы представления автоматных программ:

- стиль языка Фортран, используемый в представленном подходе;
- графические языки;
- кодирование по Switch-технологии [20];
- использование рекурсивных вызовов вместо операторов перехода;
- предметно-ориентированные языки (DSL).

Язык автоматного программирования сочетает в себе стили языков Си и Фортран. Подобный стиль встречается также в программах ядра ОС Linux[1].

Использование меток в TLA+ [22] внешне похоже на управляющие состояния в автоматном программировании. Декларировано, что метки всего лишь фиксируют участки атомарности в спецификации TLA+.

Алгоритм на графическом языке Дракон [23] эргономично представляет гиперграфовую композицию автоматной программы: ветка алгоритма непосредственно соответствует сегменту кода; все переходы по именам веток реализуются по контуру алгоритма.

В Switch-технологии [20] управляющие состояния становятся значениями некоторой переменной, а сегменты кода — альтернативами оператора switch. При таком преобразовании программа становится длиннее, сложнее и менее эффективной. В этом можно убедиться, сравнив программу LightControl (подразд. 4.3) с ее преобразованной версией в разд. 5. Для языков, не содержащих оператора перехода, это лучший способ кодирования автоматных программ.

Сегмент кода автоматной программы можно представить в виде рекурсивной функции. Аргументами функции являются переменные состояния програм-

[1] Например, https://elixir.bootlin.com/linux/v5.11.8/source/kernel/bpf/hashtab.c#L1467

мы. Функция вычисляет набор значений модифицированного состояния программы при завершении исполнения сегмента. Оператор перехода на другой сегмент заменяется вызовом рекурсивной функции для соответствующего сегмента. Преобразованная программа длиннее и сложнее, чем в предыдущих способах кодирования автоматных программ. Данный способ применяется в алгебрах процессов CCS [27], CSP [28] и др., а также в функциональном языке Erlang [29], ориентированном на разработку распределенных вычислительных систем. Отметим, что гиперграфовая композиция могла бы стать другим базисом для построения алгебр процессов.

Худшим способом является кодирование автомата программы в специальном предметно-ориентированном языке (*domain-specific language*, DSL). Это, например, языки AbC [30] и Microsoft P [31, 32]. Набор элементарных конструкций в таких языках ограничен. Исполнение программ реализуется через интерпретатор. Имеется также возможность трансляции программы на язык Си.

В событийно-ориентированной (*event-driven*) парадигме [33] программа представлена в виде цикла, в котором последовательно просматривается определенный набор событий (сообщений). Для всякого события запускается соответствующий обработчик события. Автоматная программа непредставима в такой схеме, особенно когда обработка одного события реализуется в разных сегментах кода.

Заключение

В статье определен новый язык автоматного программирования на базе языка спецификаций Event-B [1]. Его использование дает возможность разработки программ по технологии автоматного программирования с применением разнообразных эффективных инструментов верификации системы Rodin [2]. Такая возможность особенно актуальна для разработки систем управления в критической инфраструктуре с высокой ценой ошибки. Наличие трудно обнаруживаемых ошибок в распределенных системах — типичное явление, что убедительно продемонстрировано в разд. 5.

Спецификация Event-B представляет собой цепочку уточнений машин. Машина определяется недетерминированной композицией событий. А событие — эквивалент простого условного оператора. В автоматном программировании кроме недетерминированной композиции допускается ряд других. Основной является гиперграфовая композиция по управляющим состояниям. Возможны также параллельная композиция процессов, объектно-ориентированная и аспектно-ориентированная композиции. Процесс может быть вызван из другого процесса. Имеются также механизм сообщений и действий со временем. Преимущества автоматного программирования продемонстрированы на известной задаче управления движением автомобилей на мосту.

Автоматную программу нетрудно переписать в спецификацию Event-B. Техника такого переписывания показана в разд. 5. Однако предварительно следует устранить вызовы процессов, вызовы программ-функций, определяемых независимо от автоматной программы, и вызовы предикатов, используемых в формулах. Вместо вызова программы-функции можно использовать ее спецификацию через before-after-предикат. Иногда трудно найти эквивалент в Event-B для последовательного оператора: A; B. Здесь можно воспользоваться автоматной трансформацией, заменяющей сегмент M: A; B #N комбинацией M: A #K и K: B #N.

Полноценная реализация системы автоматного программирования должна предусматривать генерацию формул корректности для автоматных программ в интеграции с традиционной системой верификации на базе логики Хоара [9] для фрагментов автоматных программ, относящихся к классу программ-функций. Верификация автоматных программ должна быть поддержана инструментами, которые по своим возможностям интегрируют системы Rodin [2] и Why3 [17].

Доступная авторская реализация в Rodin задачи управления движением автомобилей на мосту представлена по ссылке https://persons.iis.nsk.su/files/persons/pages/bridge.zip.

Список литературы

1. **Abrial J.-R.** Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010. 586 p.
2. **Rodin** User's Handbook. Version 2.8 / Jastram M. (editor). 2014. 184 p.
3. **Butler M. J., Körner P., Krings S., Lecomte T., Leuschel M., Mejia L.-F., Voisin L.** The First Twenty-Five Years of Industrial Use of the B-Method // Formal Methods for Industrial Critical Systems (FMICS). 2020. P. 189—209.
4. **Карнаухов Н. С., Першин Д. Ю., Шелехов В. И.** Язык предикатного программирования Р. Новосибирск: ИСИ СО РАН, 2018. 45 с. URL: http://persons.iis.nsk.su/files/persons/pages/plang14.pdf
5. **Шелехов В. И.** Язык и технология автоматного программирования // Программная инженерия. 2014. № 4. С. 3—15. URL: http://persons.iis.nsk.su/files/persons/pages/automatProg.pdf
6. **Тумуров Э. Г., Шелехов В. И.** Технология автоматного программирования на примере программы управления лифтом // Программная инженерия. 2017. Том 8, № 3. С. 99—111. URL: http://novtex.ru/prin/full/pi317_web-99-111.pdf
7. **Шелехов В. И.** Разработка автоматных программ на базе определения требований // Системная Информатика. 2015. № 1. С. 1—29. URL: http://persons.iis.nsk.su/files/persons/pages/req_tech.pdf
8. **Шелехов В. И.** Оптимизация автоматных программ методом трансформации требований // Программная инженерия. 2015. № 11. С. 3-13. URL: http://novtex.ru/prin/full/pi1115_web-3-13.pdf
9. **Hoare C. A. R.** An axiomatic basis for computer programming // Communications of the ACM. 1969. Vol. 12 (10). P. 576—585.
10. **Floyd R. W.** Assigning meanings to programs // Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science. AMS, 1967. P. 19—32.
11. **Шелехов В. И.** Классификация программ, ориентированная на технологию программирования // Программная инженерия. 2016. Том 7, № 12. С. 531—538. URL: http://novtex.ru/prin/full/pi1216_web-531-538.pdf
12. **Liskov B., Zilles S.** Programming with abstract data types // Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages. 1974. SIGPLAN Notices 9. P. 50—59.
13. **Systems** and software engineering — Life cycle processes — Requirements engineering. ISO/IEC/ IEEE 29148, 2011, 95 p.
14. **Шелехов В. И.** Разработка программы построения дерева суффиксов в технологии предикатного программирования // Препр. ИСИ СО РАН, № 115. Новосибирск, 2004. 52 с.

15. **Шелехов В. И.** Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования // Препр. ИСИ СО РАН, № 164. Новосибирск, 2012. 30 с.
16. **Owre S., Rushby J. M., Shankar N.** PVS: Specification and Verification System // CADE-11: Automated Deduction. LNCS. 1992. Vol. 607. P. 748—752.
17. **Why 3.** Where Programs Meet Provers. URL: http://why3.lri.fr
18. **The Coq Proof Assistant.** URL: https://coq.inria.fr/
19. **Шелехов В. И.** Сравнение технологий автоматного программирования и Event-B // Системная информатика. 2021. № 18. С. 53—84. URL: https://persons.iis.nsk.su/files/persons/pages/bridge.pdf
20. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. URL: http://is.ifmo.ru/books/switch/1
21. **Specification** and description language (SDL). ITU-T Recommendation Z.100 (03/93). URL: http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf
22. **Lamport L.** Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2021. 382 p.
23. **Пароджанов В. Д.** Язык ДРАКОН. Краткое описание. М., 2009. 124 с. URL: http://drakon.su/_media/biblioteka/drakondescription.pdf
24. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. 2-е изд. СПб.: Питер, 2020. 176 с.
25. **Hoang T. S., Dghaym D., Snook C., Butler M.** A Composition Mechanism for Refinement-Based Methods // 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), 2017. P. 100—109.
26. **Abrial J.-R.** The B-book — assigning programs to meanings. Cambridge University Press, 2005. 816 p.
27. **Milner R.** Communication and Concurrency. International Series in Computer Science. Prentice Hall, 1989. 260 p.
28. **Hoare C. A. R.** Communicating Sequential Processes. Prentice Hall International. 1985. 260 p.
29. **Larson J.** Erlang for concurrent programming // ACM Queue. 2008. No. 5. P. 18—23.
30. **Nicola R. D., Duong, T., Inverso O.** Verifying AbC Specifications via Emulation // Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles, ISoLA 2020, Part II. LNCS 12477. P. 261—279.
31. **Liu P., Wahl T., Lal A.** Verifying Asynchronous Event-Driven Programs Using Partial Abstract Transformers // Computer Aided Verification. 2019. LNCS 11562. P. 386—404.
32. **Desai A., Qadeer S., Seshia S. A.** Programming safe robotics systems: Challenges and ad-vances. // Leveraging Applications of Formal Methods, Verification and Validation. Ver-ification. 2018. LNCS 11245. P. 103—119.
33. **Ferg S.** Event-Driven Programming: Introduction, Tutorial, History. SourceForge, 2006. 59 p.

Automata-based Software Engineering with Event-B

V. I. Shelekhov, vshel@iis.nsk.su, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation

Corresponding author:

Shelekhov Vladimir I., Head of Laboratory, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation
E-mail: vshel@iis.nsk.su

Received on February 13, 2022
Accepted on February 24, 2022

A new automata-based programming language has been built by extending the Event-B specification language. When developing models in Event-B, it becomes possible to use automata-based methods in addition to the popular refinement method. Automata-based software engineering, supported by deductive verification in Event-B, can be successfully used for the development of control systems in critical infrastructure with a high cost of error.

A model of the Event-B specification in the automata-based programming language is constructed. The Event-B specification is a chain of machine refinements. The machine is defined by a non-deterministic composition of events. An event is the equivalent of a simple conditional statement without else branch. In automata-based software engineering, in addition to non-deterministic composition, a number of other compositions are allowed. The main one is a hypergraphic composition with respect to control states. Parallel process composition, object-oriented and aspect-oriented compositions are also possible. A process can be called from another process. It is possible to send and receive messages. There are different time actions. It is not difficult to rewrite an automata-based program into the Event-B specification.

The automata-based software engineering with Event-B is demonstrated by the example of the problem of traffic control on the bridge from the Event-B system manual. A simpler solution with verification in the Rodin tool is proposed. The effectiveness of Event-B verification methods is confirmed by finding three non-trivial errors in our solution.

Keywords: *automata-based engineering, Event-B, refinement, requirements, program transformation, deductive verification, functional programming*

For citation:

Shelekhov V. I. Automata-based Software Engineering with Event-B, *Programmnaya Ingeneria,* 2022, vol. 13, no. 4, pp. 155—167.

DOI: 10.17587/prin.13.155-167

References

1. **Abrial J.-R.** *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, 2010, 586 p.
2. ***Rodin*** *User's Handbook*. Version 2.8, Jastram M. (editor), 2014, 184 p.
3. **Butler M. J., Körner P., Krings S., Lecomte T., Leuschel M., Mejia L.-F., Voisin L.** The First Twenty-Five Years of Industrial Use of the B-Method, *Formal Methods for Industrial Critical Systems* (FMICS), 2020, pp. 189—209.
4. **Kharnaukhov N., Perchine D., Shelekhov V.** *The predicate programming language P,* Novosibirsk, ISI SB RAN, 2018, 45 p. (in Russian).
5. **Shelekhov V. I.** Automata-based software engineering: the language and development methods. *Programmnaya Ingeneria,* 2014, no. 4, pp. 3—15 (in Russian).
6. **Tumurov E. G., Shelekhov V. I.** Applying Automata-based Software Engineering for the Lift Control Program, *Programmnaya Ingeneria,* 2017, vol. 8, no. 4, pp. 99—111 (in Russian).
7. **Shelekhov V. I.** Automata-based programming on the base of requirements specification, *Sistemnaya Informatika*, 2015, no. 1, pp. 1—29 (in Russian).
8. **Shelekhov V. I.** Automata-based Program Optimization by Applying Requirement Transformations, *Programmnaya Ingeneria*, 2015, no. 11, pp. 3—13 (in Russian).
9. **Hoare C. A. R.** An axiomatic basis for computer programming, *Communications of the ACM,* 1969, vol. 12 (10). pp. 576—585.
10. **Floyd R. W.** Assigning meanings to programs, *Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science*, AMS, 1967, pp. 19—32.
11. **Shelekhov V. I.** Program Classification in Software Engineering, *Programmnaya Ingeneria,* 2016, vol. 7, no. 12, pp. 531—538 (in Russian).
12. **Liskov B., Zilles S.** Programming with abstract data types, *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages,* SIGPLAN Notices. 9, 1974, pp. 50—59.
13. **Systems** and software engineering — Life cycle processes — Requirements engineering. ISO/IEC/ IEEE 29148, 2011, 95 p.
14. **Shelekhov V. I.** Development of a program for building a suffix tree in the predicate software engineering. Preprint, no. 115, Novosibirsk, ISI SB RAN, 2004. 52 p. (in Russian).
15. **Shelekhov V.** Verification of the heapsort predicate program using inverse transformations, *Sistemnaya informatika*, 2020, no. 16, pp. 75—102 (in Russian).
16. **Owre S., Rushby J. M., Shankar N.** PVS: Specification and Verification System, *CADE-11: Automated Deduction. LNCS,* 1992, vol. 607, pp. 748—752.
17. **Why 3.** Where Programs Meet Provers, available at: http://why3.lri.fr
18. **The Coq** Proof Assistant, available at: http://coq.inria.fr
19. **Shelekhov V. I.** Comparison of automata-based engineering method and Event-B modeling method, *Sistemnaya informatika*, 2021, no. 18, pp. 53—84 (in Russian).
20. **Shalyto A. A.** *SWITCH-technology. Algorithmic and Programming Methods in Solution of Logic Control Problems*, St. Petersburg, Nauka, 1998, 628 p. (in Russian).
21. **Specification** and description language (SDL). ITU-T Recommendation Z.100 (03/93), available at: http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf
22. **Lamport L.** *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2021, 382 p.
23. **Parondzanov V. D.** *DRAKON language,* Short description, Moscow, 2009, 124 p. (in Russian).
24. **Polykarpova N. I., Shalyto A. A.** *Automata-base programming.* Second edition, St. Petersburg, Piter, 2020, 176 p. (in Russian).
25. **Hoang T. S., Dghaym D., Snook C., Butler M.** A Composition Mechanism for Refinement-Based Methods, *22nd International Conference on Engineering of Complex Computer Systems* (ICECCS), 2017, pp. 100—109.
26. **Abrial J.-R.** *The B-book — assigning programs to meanings*, Cambridge University Press, 2005, 816 p.
27. **Milner R.** *Communication and Concurrency,* International Series in Computer Science, Prentice Hall, 1989, 260 p.
28. **Hoare C. A. R.** *Communicating Sequential Processes*, Prentice Hall International, 1985, 260 p.
29. **Larson J.** Erlang for concurrent programming, *ACM Queue*, 2008, no. 5, pp. 18—23.
30. **Nicola R. D., Duong, T., Inverso O.** Verifying AbC Specifications via Emulation, *Leveraging Applications of Formal Methods*, Verification and Validation: Engineering Principles, ISoLA 2020, Part II, LNCS 12477, pp. 261—279.
31. **Liu P., Wahl T., Lal A.** Verifying Asynchronous Event-Driven Programs Using Partial Abstract Transformers, *Computer Aided Verification*, LNCS 11562, 2019, pp. 386—404.
32. **Desai A., Qadeer S., Seshia S. A.** Programming safe robotics systems: Challenges and advances. *Leveraging Applications of Formal Methods, Verification and Validation. Verification*, 2018, LNCS 11245, pp. 103—119.
33. **Ferg S**. *Event-Driven Programming: Introduction*, Tutorial, History, SourceForge, 2006, 59 p.

ИНФОРМАЦИЯ

Всероссийская научно-техническая конференция

"Многопроцессорные вычислительные и управляющие системы"

МВУС-2022

27—30 июня 2022 г.
Таганрог, Ростовская область

Конференция посвящена 100-летию со дня рождения выдающегося российского ученого, Героя социалистического труда, академика РАН А. В. Каляева и развитию его научных идей.

Целью конференции является представление достижений российских ученых в развитии фундаментальных исследований и прикладных разработок в области многопроцессорных вычислительных и управляющих систем.

Направления работы

- Академик РАН А. В. Каляев и его научная школа
- Архитектура многопроцессорных вычислительных и управляющих систем
- Математическое и системное программное обеспечение многопроцессорных вычислительных и управляющих систем
- Реконфигурируемые вычислительные системы
- Проблемно-ориентированные и встраиваемые вычислительные системы
- Многопроцессорные вычислительные и управляющие системы в системах искусственного интеллекта

Подробности: **https://conf.mvs.sfedu.ru/**

УДК 004.054 DOI: 10.17587/prin.13.168-177

А. Н. Годунов, канд. физ.-мат. наук, зав. отделом, nkag@niisi.ras.ru,
И. И. Хоменков, вед. инж., nkigor@niisi.ras.ru, **В. Г. Щепков**, вед. инж., nkvs@niisi.ras.ru,
Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук (ФНЦ НИИСИ РАН), Москва,
А. В. Хорошилов, канд. физ.-мат. наук, вед. науч. сотр., khoroshilov@ispras.ru, Институт системного программирования им. В. П. Иванникова Российской академии наук (ИСП РАН), Москва

Конфигурируемая тестовая система для ОСРВ семейства Багет

Описана тестовая система, предназначенная для проверки работоспособности операционных систем реального времени, которая была разработана и используется в НИИСИ РАН для автоматизации процесса тестирования программного обеспечения. Если большинство тестовых систем проектируются, в первую очередь, с ориентацией на автоматизацию сборки, проведение тестирования, а также сбора, обработки и визуализации полученных результатов, то в рассматриваемой системе помимо этого предусмотрены специальные возможности для упрощения отладки и исправления обнаруживаемых проблем, что позволяет сократить усилия разработчиков операционной системы реального времени на анализ результатов тестирования.

Ключевые слова: *операционная система реального времени (ОСРВ), тестирование программного обеспечения, автоматизация программирования, кросс-платформенная разработка, языки программирования C, C++, POSIX, ARINC-653*

Введение

При разработке операционных систем необходимо проводить тестирование для проверки надежности и соответствия стандартам. И как следовало ожидать, существует специализированное программное обеспечение (ПО) для автоматизации тестирования Unix-подобных операционных систем, например, Avocado [1], LAVA [2], Linux Test Project [3], Linux Distribution Checker [4], Open POSIX Test Suite [5], UnixBench [6] и т. д. Но использование таких готовых программных систем не всегда удобно, так как они либо содержат только узкоспециализированные наборы тестов, либо поддерживают только определенную аппаратуру, либо не содержат гибкой системы конфигурирования. Так или иначе, требуются большие трудозатраты на адаптацию готовой тестовой системы для тестирования нового семейства операционных систем. Универсальные системы автоматизации тестирования, такие как BuildBot [7], Jenkins [8], GitLab CI [9] и др., не учитывают особенности тестирования операционных систем, связанные с необходимостью выполнения тестируемой системы на аппаратных устройствах или их эмуляторах и также требуют существенной доработки. Поэтому в НИИСИ РАН была разработана собственная оригинальная тестовая система (ТС) для проверки работоспособности операционных систем реального времени (ОСРВ) семейства Багет [10], которые разрабатываются в НИИСИ РАН для применения во встраиваемых системах.

1. Назначение и возможности

Тестирование является неотъемлемой частью разработки ПО. Для достижения высокого качества ПО требуется, как правило, выполнение большого числа тестов. Их разработка сравнима по трудоемкости с разработкой собственно тестируемого ПО. Отметим, что тесты приходится запускать многократно в различных конфигурациях на различных устройствах, как в процессе разработки нового ПО, так и при выпуске обновлений уже существующего.

Время выполнения тестов в случае сложного ПО может быть значительным (вплоть до нескольких суток), запуск тестов (указание значений параметров и создание ПО нужной конфигурации) и анализ результатов требуют много времени и высокой квалификации тестировщиков. В силу этого возникает потребность в автоматизации процесса тестирования.

С этой целью в НИИСИ РАН в сотрудничестве с ИСП РАН была разработана ТС, которая уже несколько лет используется при разработке ОСРВ семейства Багет.

При разработке ТС ставились задачи создать удобный инструмент тестирования как для тестировщи-

ков, так и для программистов, снизить трудоемкость и сократить время тестирования. Тестировщики, с одной стороны, нуждаются в удобном средстве для запуска большого числа разнообразных тестов без трудоемкого указания всего набора параметров ОСРВ. С другой стороны, программистам нужно иметь возможность быстро найти тест, завершившийся с ошибкой, и определить условия, в которых он выполнялся. Если такая ошибка была обнаружена ранее и зарегистрирована в системе отслеживания ошибок, нужно найти ее описание, обсуждение и текущее положение дел по ее устранению.

При создании прикладного ПО для ОСРВ используется кросс-платформенная разработка, когда процесс разработки программного кода и исполнение готовых программ происходят на компьютерах с разной архитектурой. Компьютер, на котором ведется разработка, называется инструментальной ЭВМ (ИЭВМ), а компьютер, на котором выполняется ПО, — целевой ЭВМ (ЦЭВМ). После завершения цикла разработки ПО может работать на ЦЭВМ автономно. В нашем случае ИЭВМ — это компьютер с процессором Intel, который работает под управлением операционной системы Linux, а ЦЭВМ — это компьютер с микропроцессором архитектуры MIPS под управлением ОСРВ семейства Багет.

Тестовая система имеет клиент-серверную архитектуру. Сервер ТС (приложение, написанное на языке JAVA) устанавливается на ИЭВМ. В качестве клиентов используются web-браузеры, которые с помощью удобного интерфейса позволяют запускать тесты ОСРВ на разных ЦЭВМ и анализировать результаты тестирования.

Тестовая система обеспечивает автоматизацию следующих рутинных процессов тестирования ОСРВ:

- установку новых версий ОСРВ (а также удаление или переустановку старых версий);
- конфигурирование ОСРВ, т. е. настройку параметров для конкретной ЦЭВМ (сетевые адреса, имена терминалов, каталоги на сервере и т. д.);
- сборку исполняемых образов ОСРВ (с различными тестами, включенными в образ);
- загрузку на ЦЭВМ исполняемых образов ОСРВ и поочередный запуск тестов;
- сбор и хранение на сервере протоколов выполнения тестов и их комплексный анализ;
- представление результатов тестирования в удобной форме, их архивирование на сервере и визуализацию по запросу оператора.

Процесс полного тестирования ОСРВ (от запуска тестов до получения результатов) может быть длительным. Для экономии времени предусмотрены различные сценарии запуска тестов. Возможен запуск не только всех тестов сразу на всем спектре оборудования, но и запуск выбранных групп тестов на выбранных архитектурах, а также запуск одиночных тестов, что удобно в процессе разработки и отладки ОСРВ. Имеются также другие способы управления объемом тестирования, например, можно запустить выполнение только тех тестов, которые не были выполнены или были выполнены с ошибкой ранее. Такой подход к тестированию позволяет экономить много времени и ресурсов за счет автоматизации рутинных операций и более эффективного использования оборудования и рабочего времени. Тестовая система может выполнять тесты одновременно на нескольких ЦЭВМ, максимально загружая оборудование и экономя время.

Также существует возможность запускать тесты ОСРВ традиционным способом (в ручном режиме). Такой способ тестирования используется, когда нужно провести серию однотипных официальных испытаний ОСРВ. Для автоматизации этого процесса в ТС было разработано приложение на языке JAVA, которое использует интерфейс командной строки вместо графического web-интерфейса. Оно позволяет выполнять все рутинные процедуры по конфигурированию, сборке, запуску тестов и сборке протоколов тестирования. Но анализ протоколов выполняется вручную (полуавтоматический режим).

Таким образом можно сказать, что ТС является полезным и эффективным инструментом, который позволяет:

- повысить качество конечного продукта (ОСРВ) за счет возможности выполнения большого числа тестов на большой номенклатуре оборудования;
- снизить суммарное время проведения полного тестирования за счет автоматизации всех подготовительных операций, распараллеливания тестирования и возможности непрерывной круглосуточной работы;
- снизить число ошибок при тестировании, связанных с "человеческим фактором" (невнимательность, забывчивость, торопливость и т. д.), и, как следствие, повысить достоверность результатов тестирования и сократить время тестирования;
- уменьшить время на поиск ошибок за счет автоматического анализа протоколов тестирования и возможности быстро повторить тест, завершившийся с ошибкой;
- архивировать результаты тестирования на сервере, хранить их сколь угодно долго и предоставлять программистам по первому требованию со своих рабочих мест;
- использовать web-интерфейс для максимального удобства работы с ТС.

2. Используемая аппаратура

2.1. Состав стенда

На рис. 1 представлены фрагменты реального стенда, на котором происходит тестирование ОСРВ в НИИСИ РАН.

Стенд для тестирования и разработки ОСРВ состоит из одной ИЭВМ (высокопроизводительный выделенный сервер, на котором установлена ТС), совокупности ЦЭВМ разных типов и инфраструктуры, необходимой для их взаимодействия:

- внешняя сеть позволяет нескольким программистам одновременно работать удаленно с ТС и по-

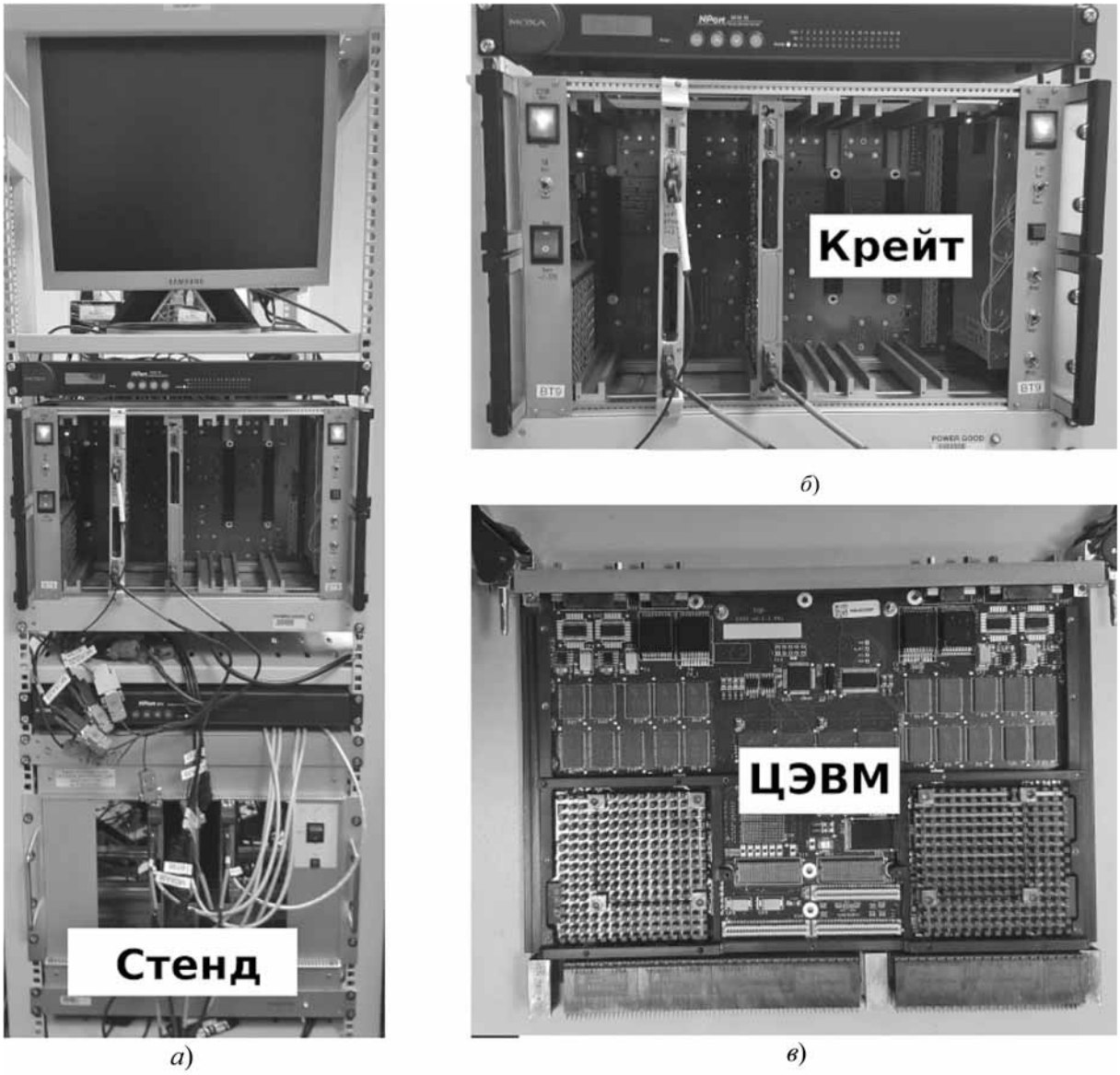

Рис. 1. Оборудование ТС:
а — фрагмент тестового стенда; *б* — крейт; *в* — ЦЭВМ

лучать результаты тестирования со своих рабочих мест;

- внутренняя сеть используется для организации взаимодействия между различными компонентами тестового стенда по сети Ethernet;
- коммутаторы последовательных портов RS-232 используются для подключения к ЦЭВМ через терминальный интерфейс, который обеспечивает реализацию интерфейса управляющей командной строки, а также получение текстовой информации с результатами выполнения тестов; взаимодействие между коммутаторами портов RS-232 и ИЭВМ осуществляется по сети Ethernet;
- генератор аппаратных сигналов RESET на перезагрузку ЦЭВМ; выдачей сигналов на перезагрузку управляет сервер ТС по сети Ethernet;
- графические мониторы, графические манипуляторы и клавиатуры используются в качестве графических терминалов ЦЭВМ, когда разрабатываются приложения для ОСРВ, которые требуют интерактивного взаимодействия с человеком, например, рабочее место оператора.

В состав стенда в настоящее время входят несколько десятков ЦЭВМ с различными типами процессоров, разработанных в НИИСИ РАН, таких как КОМДИВ32, КОМДИВ64, КОМДИВ64-SMP, КОМДИВ64-RIO, КОМДИВ128-RIO [11]. Эти ЦЭВМ (модули) вставляют в крейты (корпус с общей шиной) по несколько штук. Крейты, в свою очередь, монтируются в индустриальные стойки с необходимой вспомогательной инфраструктурой. Стенд для тестирования и разработки ОСРВ состоит из нескольких стоек с оборудованием, которые в значительной степени не зависят друг от друга. Такой подход позволяет легко масштабировать стенд для подключения новых типов ЦЭВМ путем добавления новых стоек или новых крейтов.

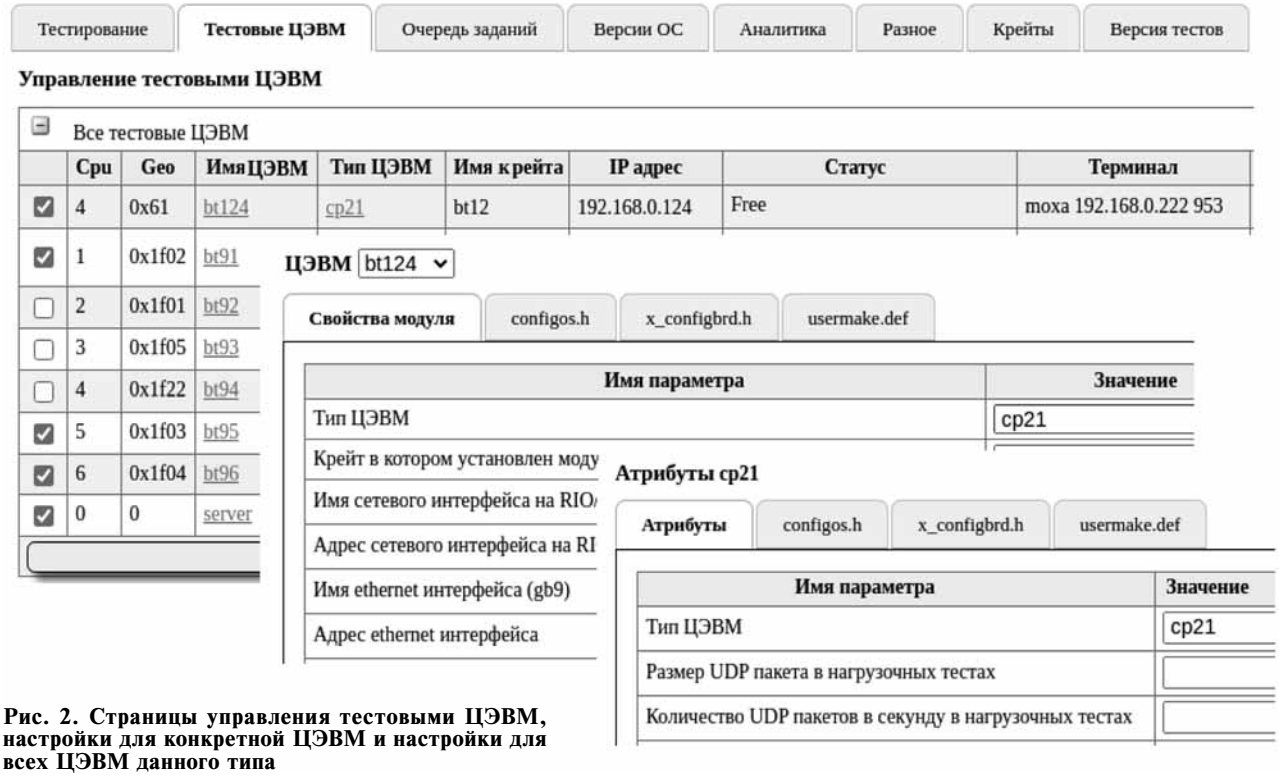

Рис. 2. Страницы управления тестовыми ЦЭВМ, настройки для конкретной ЦЭВМ и настройки для всех ЦЭВМ данного типа

2.2. Конфигурирование ЦЭВМ

Программный сервер ТС работает на ИЭВМ и взаимодействует со всем спектром ЦЭВМ. Для этого он использует формальное описание всех имеющихся на стенде ЦЭВМ. Страница "Управление тестовыми ЦЭВМ" сервера содержит сводную таблицу всех ЦЭВМ, зарегистрированных в ТС (фрагмент таблицы приведен на рис. 2).

Каждая строка в сводной таблице относится к одной ЦЭВМ (или к одному процессору в многопроцессорных модулях), а в полях этой строки перечислены наиболее важные параметры, которые используются ТС для взаимодействия с этой конкретной ЦЭВМ. В этой таблице можно добавлять новые ЦЭВМ (нажав кнопку внизу таблицы) или редактировать настройки уже существующих. Настройки ЦЭВМ делятся на две группы: настройки для конкретной ЦЭВМ и настройки для всех ЦЭВМ данного типа.

При нажатии на ссылку в столбце "Имя" сводной таблицы происходит переход на страницу, где указываются атрибуты (параметры) только для одной конкретной ЦЭВМ (средняя таблица на рис. 2).

Перейдя по ссылке в столбце "Тип ЦЭВМ" сводной таблицы, можно попасть на страницу атрибутов этого типа ЦЭВМ (нижняя таблица на рис. 2). Эти настройки применяются для всех ЦЭВМ такого типа.

3. Программное обеспечение

3.1. Установка и удаление версий ОСРВ в ТС

Дистрибутив ОСРВ содержит базовый набор программ, сценариев и библиотек, необходимых для построения исполняемых образов. Для установки ОСРВ на компьютер используется сценарий, написанный на языке командного интерпретатора Linux. Установленная таким образом ОСРВ содержит все необходимое для поддержки конкретной микропроцессорной архитектуры, но не содержит драйверов периферийных устройств, которые могут различаться для разных типов ЦЭВМ. Поэтому, кроме установки дистрибутива ОСРВ нужно установить один или несколько пакетов поддержки модулей (ППМ), которые содержат драйверы для конкретных типов ЦЭВМ.

Тестовая система имеет доступ к каталогам, куда разработчики помещают дистрибутивы версий ОСРВ, и периодически проверяет содержимое этих каталогов. Страница ТС "Версии ОС" позволяет устанавливать и удалять версии ОСРВ и ППМ с помощью следующих вкладок:

- **Установленные версии** — таблица со списком уже установленных в ТС версий ОСРВ, а также версий ППМ к ним;
- **Неустановленные версии** — таблица версий ОСРВ, которые были созданы, но еще не были установлены в ТС;
- **Архивные версии** — дополнительный раздел, который аналогичен разделу "Установленные версии"; здесь перечислены версии ОСРВ, разработка которых была завершена.

После успешной установки информация о версии перемещается из раздела "Неустановленные версии" в раздел "Установленные версии". Для того чтобы версия ОСРВ попала список архивных версий, нужно присвоить ей соответствующий статус.

При установке ОСРВ создаются один или несколько каталогов, предназначенных для работы программиста, которые называют целевыми. Они содержат все необходимые компоненты для построения исполняемого образа ОСРВ для конкретного типа ЦЭВМ (по одному каталогу на каждый тип ЦЭВМ). В целевом каталоге размещаются исходные тексты программ на языках С или С++. Построение исполняемого образа начинается с запуска конфигуратора ОСРВ. Конфигуратор позволяет включить тексты программ в создаваемый исполняемый образ ОСРВ, а также задать конфигурационные параметры ОСРВ (размер памяти, сетевые адреса и т. д.). По завершению конфигурирования приводится компиляция и сборка исполняемого образа ОСРВ в виде файла, который можно загрузить и выполнить на ЦЭВМ.

3.2. Структура тестов ОСРВ

Все тесты ОСРВ сгруппированы в подкаталоги по принципу проверки какой-нибудь одной подсистемы ОСРВ. Тестовым набором авторы называют все тесты одного такого подкаталога. Каждый тестовый набор состоит из отдельных тестов, которые оформлены в виде файлов на языке С (С++). Обычно все тесты тестового набора помещаются в один исполняемый образ ОСРВ и выполняются последовательно, хотя есть и исключения из этого правила. Иногда случаются ситуации, когда ошибка, возникшая при выполнении одного теста, может повлиять на выполнение последующих тестов (например, ошибочное затирание памяти). Если необходимо исключить возможное влияние одного теста на другой, то можно построить исполняемый образ ОСРВ только с одним тестом из тестового набора и выполнить только его (такой режим тестирования требует существенно больше времени).

Перечислим наиболее важные тестовые наборы:

- тесты для проверки ОСРВ на соответствие стандарту ARINC-653;
- тесты для проверки ОСРВ на соответствие стандарту POSIX;
- тесты для проверки математических функций (sin, cos и т. д.) на точность вычисления результата с учетом различных режимов округления чисел с плавающей точкой;
- тесты для проверки ОСРВ (семейства ОС4000) на соответствие стандарту языка С++;
- тесты производительности, измеряющие скорость выполнения различных функций ОСРВ (при этом не проверяется корректность выполнения этих функций).

Последний пункт требует дополнительных пояснений. Тесты производительности (*benchmarks*) не являются корректными тестами, но позволяют оценивать временные характеристики ОСРВ и сравнивать между собой быстродействие различных микропроцессорных модулей и различных версий ОСРВ (это можно сделать на странице "Аналитика", описанной ниже). Дополнительно они являются средством обнаружения "скрытых" проблем в ПО. Например, если тест вычисления какой-либо математической функции дает верный результат (что проверяется математическими тестами), но время ее выполнения сильно увеличивается по сравнению с предыдущими версиями ОСРВ (что проверяют тесты производительности), можно предположить, что в ходе выполнения этой функции происходят исключительные ситуации (*exceptions*). Эти исключительные ситуации корректно обрабатываются соответствующими обработчиками, но эти обработчики требуют дополнительного процессорного времени, что может быть критично в системах реального времени. Тесты производительности могут также являться средством профилирования ПО (сбором статистики о времени выполнения какой-либо функции и о числе вызовов этой функции) и помогать оптимизировать программный код.

3.3. Управление тестовыми наборами

Перед использованием тестов соответствующие тестовые наборы должны быть предварительно зарегистрированы в ТС. Для этих целей служит страница "Управление тестами", представленная на рис. 3.

Эта страница содержит таблицу, где перечислены все тестовые наборы. Она также позволяет управлять ими: добавлять, удалять, редактировать и т. д. При нажатии на ссылку в столбце "Имя тестового набора" происходит переход на страницу конфигурирования для данного тестового набора. Там указывается, на каких целевых платформах и с какими параметрами этот тестовый набор может запускаться. Кроме того, на странице содержится полный список тестов в этом наборе. Это позволяет сравнить список реально выполненных тестов с эталонным списком и обнаружить невыполненные или пропущенные по каким-то причинам тесты.

3.4. Построение исполняемых образов ОСРВ

Важным этапом при построении исполняемого образа ОСРВ является процесс конфигурирования. Здесь задаются конкретные параметры и структура будущего образа, например, сетевые адреса, каталоги с файлами, пользовательские программы и т. д. Все эти настройки сохраняются в виде конфигурационных файлов.

Для автоматизации процесса конфигурирования в ТС заранее заготовлены фрагменты конфигурационных файлов, относящиеся к различным режимам работы ОСРВ, ЦЭВМ и тестовых наборов. Эти фрагменты конфигурационных файлов хранятся в отдельном каталоге ТС, который имеет сложную систему подкаталогов и названий файлов, позволяющую быстро получить доступ к нужным фрагментам конфигурации ОСРВ.

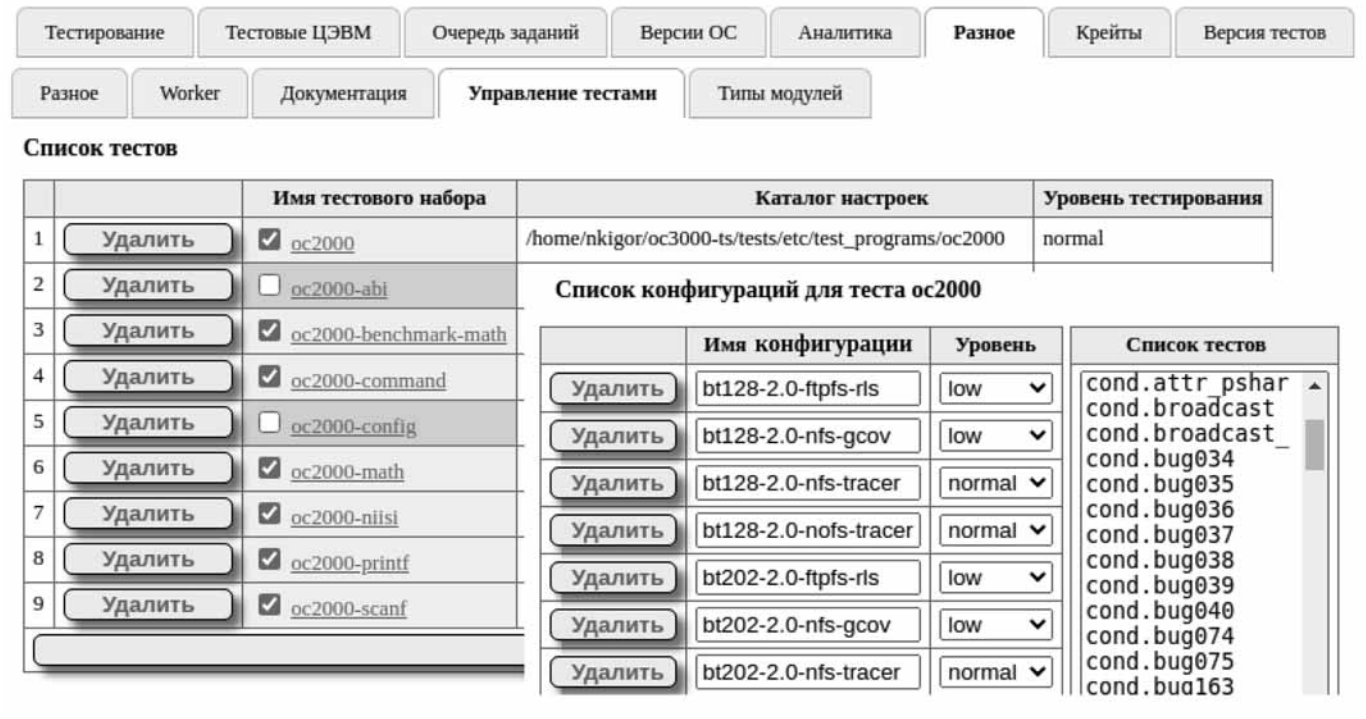

Рис. 3. Страницы управления и конфигурирования тестов

Для того чтобы создать конкретную конфигурацию для тестирования ОСРВ, программист (при первоначальной настройке ТС) задает ее имя следующим образом: имя конфигурации должно состоять из цепочки ключевых слов, соединенных между собой дефисами. Ключевые слова в имени — это названия отдельных файлов с фрагментами конфигураций, которые находятся в подкаталогах, упомянутых выше. Эти файлы будут добавляться к конфигурационным файлам ОСРВ в процессе автоматического конфигурирования.

Имя конфигурации можно увидеть в первом столбце таблицы тестирования (рис. 4, см. вторую сторону обложки). Например, на рис. 4 открыта страница тестирования в ТС, выбраны версия ОСРВ oc3000-3.52.215 и тестовый набор oc3000-math в конфигурации bt206-3.0-nfs-dbg. Для определенности укажем процессорный модуль bt61 в качестве ЦЭВМ. Запустить процесс тестирования ОСРВ в данной конфигурации можно, нажав кнопку "Выполнить тесты" в сводной таблице тестирования (рис. 4, см. вторую сторону обложки) в строке, которая начинается с названия нашей конфигурации и относится к выбранному тестовому набору. При этом ТС будет знать, что нужно использовать следующую конфигурационную информацию для построения исполняемого образа ОСРВ:

- версия ОСРВ — oc3000-3.52.215 (название каталога с установленной ОСРВ);
- тестовый набор — oc3000-math (название каталога с тестовым набором для тестирования математических функций);
- микропроцессорная архитектура ЦЭВМ — bt206 (название целевого каталога для ЦЭВМ типа bt206 с микропроцессором RM7000);
- семейство ОСРВ — 3.0;
- тип файловой системы — nfs (подключить сетевую файловую систему для чтения файлов эталонных данных с ИЭВМ);
- версия библиотек ОСРВ — dbg (использовать отладочный набор библиотек и ключей компилятора);
- конкретная ЦЭВМ — bt61 (использовать сетевой адрес 192.168.0.61 и т. д.).

Конфигурирование и сборка исполняемого образа ОСРВ проводятся следующим образом. Сначала отдельно копируется целевой каталог ОСРВ, затем формируются конфигурационные файлы ОСРВ путем копирования фрагментов конфигурации из разных файлов. После окончания конфигурирования будет выполнена процедура сборки исполняемого образа ОСРВ (компиляция всех исходных текстов проекта и создание исполняемого образа системы).

4. Тестирование

4.1. Процесс тестирования ОСРВ

Настройка и конфигурирование самой ТС проводятся после ее установки на сервер, а также по мере необходимости (например, в случае подключения нового оборудования к стенду).

Использование ТС предполагает выполнение следующих действий. На сервер устанавливаются

версия ОСРВ для тестирования и набор соответствующих ППМ для разных модулей. Затем проводятся конфигурирование и сборка исполняемых образов ОСРВ. На следующем этапе происходят загрузка исполняемых образов ОСРВ на ЦЭВМ и запуск тестов. В заключение ТС проводит анализ протоколов исполнения тестов и представление результатов в удобной для просмотра форме. Дальше тестировщик должен сам проанализировать полученные ТС данные и сделать выводы о качестве ПО и его дальнейшей судьбе.

Результаты тестирования конкретной версии ОСРВ хранятся на сервере и доступны в любое время, пока пользователь не удалит их явно.

В ходе выполнения тесты выводят на консоль протокол тестирования, который начинается с названия теста и заканчивается окончательным вердиктом:

- PASS — тест прошел успешно;
- FAIL — была обнаружена ошибка или возникли какие-то трудности при выполнении теста;
- UNTESTED — тест не был запущен в ходе тестирования в связи с невозможностью его выполнения на данной программно-аппаратной платформе.

Вердикт UNTESTED не является ошибкой тестирования. Он является следствием использования единого набора тестов для всего набора оборудования и всех конфигураций ОСРВ в целях упрощения поддержки ТС в актуальном состоянии и повышения ее надежности. Поэтому некоторые тесты запускаются только в том случае, когда для этого существуют определенные условия, например, наличие соответствующего оборудования.

При тестировании ОСРВ окончательный вердикт является совокупностью вердиктов выполнения отдельных тестовых наборов в различных режимах и конфигурациях.

4.2. Запуск тестов ОСРВ

Запустить тесты можно на странице "Тестирование". Здесь все тесты сгруппированы в виде сводной таблицы. Вся таблица может быть очень длинной и поэтому имеет два представления — развернутый и компактный (рис. 4, см. вторую сторону обложки).

Перед запуском тестов выбирается семейство ОСРВ Багет: ос2000, ос3000, ос4000 или ос4000.64 (64-разрядная версия ОСРВ ОС4000). Затем выбирается номер конкретной версии ОСРВ (например, ос3000-3.52.215). Если строчки этой таблицы содержат только названия тестовых наборов и окрашены в белый цвет, то тестирование этой версии ОСРВ еще не проводилось. Если некоторые строчки таблицы окрашены в какой-то цвет и содержат какую-либо информацию, то соответствующие тесты уже были запущены и можно посмотреть отчет о результатах тестирования.

Таблица содержит управляющие кнопки, которые позволяют запускать тесты в нужном порядке и в нужном объеме. Можно запустить на выполнение:

- все тесты на всех целевых платформах (будет заполнена вся тестовая таблица);
- все тесты на выбранной целевой платформе (будет заполнен раздел тестовой таблицы);
- группу тестов на выбранной целевой платформе в различных конфигурациях ОСРВ (будут заполнены несколько строк тестовой таблицы);
- группу тестов на выбранной целевой платформе в выбранной конфигурации ОСРВ (будет заполнена одна строка тестовой таблицы);
- только один тест ОСРВ на выбранной целевой платформе в выбранной конфигурации ОСРВ.

Для перехода в режим выполнения одиночного теста нужно нажать на ссылку в первой колонке таблицы. После этого осуществится переход на страницу выбора, где данный тестовый набор представлен в виде управляющих кнопок с названиями одиночных тестов или кнопками с названием групп тестов из данного тестового набора (рис. 5, см. вторую сторону обложки). Теперь можно запустить только один тест или небольшую группу выбранных тестов. Это удобно для поиска ошибок при отладке.

Отметим, что в ТС тесты запускаются не сразу. Вначале они попадают в очередь заданий на тестирование. Очередь формируется и движется автоматически по мере запуска тестов и освобождения оборудования для тестирования. Это позволяет запускать большое число тестов на длительный срок, например, на ночь, когда тестовый стенд свободен. На странице "Очередь заданий" можно видеть текущее состояние очереди (какие тесты сейчас выполняются и на каком оборудовании). Здесь можно остановить выполнение всех тестов и очистить очередь заданий (нажав кнопку "Остановить все тесты"), но нельзя управлять самой очередью, изменяя число и очередность заданий. Добавить или удалить отдельные тестовые наборы из очереди можно на странице "Тестирование", нажав соответствующие кнопки. Алгоритм работы очереди заданий на тестирование не предусматривает никаких приоритетов в выполнении тестов (очередь движется линейно), но при этом может запускать параллельное выполнение одного задания сразу на нескольких модулях одновременно. Например, если запустили один тестовый набор с математическими тестами и есть шесть свободных, подходящих по типу процессорных модулей, то ТС запустит сразу шесть тестов из этого тестового набора (один процессор будет выполнять тестирование функции sin, а другой — функции cos и т. д.). По мере окончания тестирования очередной математической функции каждый из шести процессорных модулей будет получать новую, таким образом сильно ускоряется процесс тестирования. В логику работы очереди заданий также включено слежение за временем выполнения тестов. Если тест тратит больше отведенного ему времени, то его выполнение прекращается, тест считается неудачным, а ТС переходит к выполнению следующего задания в очереди. Таким образом решается проблема "зависания" тестов.

По завершению тестирования (возможно, через несколько часов) можно получить все результаты тестов. Это таблица (рис. 4, см. вторую сторону обложки), где строки содержат названия тестовых наборов и результаты тестирования. Строчки итоговой таблицы раскрашены в разные цвета. Цвет позволяет быстро оценить результаты тестирования. Если строчка таблицы окрашена в зеленый цвет, то результаты всех тестов соответствующей группы положительные (в идеале вся таблица должна быть зеленого цвета), а если строчка красная, то результат хотя бы одного теста отрицательный. Возможен также фиолетовый цвет, когда тесты ОСРВ закончились с ошибками, но все эти ошибки уже известны, зарегистрированы и находятся в стадии устранения. Желтый цвет означает, что все результаты тестов этой группы были положительные (не было отрицательных), но число положительных результатов не совпадает с общим ожидаемым числом тестов этой группы. Другими словами, какие-то тесты не выполнились (нет названия теста в протоколе тестирования, нет окончательного вердикта за отведенное время, поэтому ТС не может автоматически квалифицировать ситуацию) и программисту следует разобраться с этим тестом самостоятельно. Такое возможно, например, если при старте теста возникла исключительная ситуация (*exception*), которая не может быть корректно обработана, и операционная система останавливает выполнение этого теста, т. е. получен отрицательный результат для этого теста, но косвенным образом, как отсутствие какой-либо информации о нем. Розовый цвет говорит, что ошибки возникли еще до запуска тестов на этапе компиляции или тестирование ОСРВ прервалось по неизвестной причине. Например, какой-то крейт был выключен до завершения тестов. Напомним, что белый цвет строк в таблице говорит о том, что запуск соответствующей группы тестов еще не проводился.

Надписи в этих строках, также как и цвет сообщают о результатах тестирования, например, "Нет проблем" означает, что тесты прошли успешно, а "Ошибок *N*" означает, что в ходе тестирования было обнаружено *N* ошибок.

В таблицу тестирования внесены лишь самые краткие сведения о результатах выполнения тестов: сколько тестов было запущено, сколько выполнено и сколько было ошибок. Чтобы получить подробные протоколы о результатах выполнения группы тестов нужно нажать на ссылку в строке таблицы с количеством ошибок (например, "Ошибок: 1") — будет выведена страница с подробным отчетом, которая построена на базе анализа результатов, полученных с ЦЭВМ. Эта страница является результатом обработки всех протоколов выполнения данного тестового набора. Она содержит структурированную информацию о запуске тестов и результатах их выполнения. В начале страницы расположена информация о конфигурации образа ОСРВ, времени начала и окончания тестирования, краткий список тестов с результатами выполнения (тесты с ошибками выделяются красным цветом). Затем идет подробный отчет по выполнению каждого теста в отдельности. Это дает возможность получать детальную информацию об ошибках максимально быстро без утомительного просматривания множества протоколов. Если какая-то ошибка была обнаружена раннее, то можно найти ссылку на запись в базе данных отслеживания ошибок (*Mantis Bug Tracker*) и узнать текущее состояние дел по ее устранению.

4.3. Сравнение результатов

Вкладка "Аналитика" позволяет сравнивать результаты тестирования версий ОСРВ по некоторому выбранному критерию.

Сравнивать можно время выполнения тестов на разных версиях ОСРВ. Это позволяет находить ошибки в ОСРВ если, например, производительность операционной системы сильно ухудшается. Также можно сравнивать время выполнения тестов одной и той же версии ОСРВ, но на разных микропроцессорных модулях. Это дает возможность понять, насколько один микропроцессорный модуль производительнее другого, а также находить ошибки в ОСРВ, если время выполнения меняется непропорционально на разных микропроцессорных архитектурах.

Для сравнения версий реализовано несколько алгоритмов, которые представлены в следующих закладках:

- **Сравнение типов** — в этом разделе можно получить таблицу, где сравнивается время выполнения некоторых специально выбранных функций одной и той же версии ОСРВ, но для разных микропроцессорных модулей;
- **Математические функции** — этот раздел аналогичен предыдущему, но для сравнения скорости выполнения функций одной и той же версии ОСРВ на различных микропроцессорных архитектурах выбираются только математические функции стандарта POSIX;
- **ARINC-функции** — этот раздел аналогичен предыдущим, но для сравнения скорости выполнения функций одной и той же версии ОСРВ на различных микропроцессорных архитектурах выбираются только функции стандарта ARINC;
- **Производительность по версиям** — здесь можно сравнить, насколько одна версия ОСРВ быстрее или медленнее, чем другая на одной и той же микропроцессорной архитектуре;
- **Сравнение версий** — этот раздел отличатся от предыдущих тем, что сравниваются не временные характеристики разных версий ОСРВ, а то число ошибок, которое было выявлено при тестировании; таким образом можно отслеживать, как исправляются обнаруженные в ОСРВ ошибки и увеличивается стабильность версий ОСРВ в ходе их разработки.

Результатом сравнения является текстовый файл с расширением .csv (*comma-separated values*), который можно импортировать в программу электронных таблиц, а дальше обработать информацию в нужном виде с помощью встроенных функций (например, построить графики).

4.4. Дополнительные возможности

В разделе "Разное" есть список зарегистрированных проблем в ОСРВ. База данных ошибок формируется на основании регистрации сообщений об ошибках (*ticket*) в системе регистрации и отслеживания ошибок Mantis Bug Tracker [12]. Разработчики ОСРВ регистрируют сообщения об ошибках ОСРВ в системе Mantis в ручном режиме. Для этого нужно сначала создать сообщение об ошибке (*ticket*), а система Mantis присвоит этому сообщению уникальный номер. В этом сообщении нужно описать проблему, присвоить ей статус и назначить ответственного за ее решение. В дальнейшем статус этой проблемы может меняться (в процессе работ по устранению ошибки): новая, назначенная, решенная, закрытая и т. д. Система Mantis не является частью ТС, но предоставляет возможность узнать удаленно (по сети) текущий статус сообщения об ошибке по его уникальному номеру. Эту особенность использует ТС для автоматического слежения за ошибками, возникающими в ходе тестирования ОСРВ. Для этого ТС ведет свою собственную базу данных ошибок на своем сервере и предоставляет тестировщику возможность связать конкретную ошибку, выявленную при тестировании ОСРВ, и номер зарегистрированного сообщения об этой ошибке в Mantis, нажав соответствующую кнопку при просмотре отчетов о тестировании. Впоследствии это дает возможность ТС вставлять ссылки на зарегистрированные в Mantis сообщения, относящиеся к данной проблеме, и менять вид сообщения об ошибках, базируясь на статусе сообщения об ошибке в системе Mantis. Это очень удобно, так как можно понять текущее состояние этой проблемы и что делается для ее разрешения на текущий момент. Отметим дополнительно, что если при тестировании ОСРВ были обнаружены ошибки и все эти ошибки были зарегистрированы в системе Mantis, то в сводной таблице тестирования строка с этой группой тестов закрашивается не в красный цвет, а в фиолетовый — это значит, что ошибки есть, все они известны, зарегистрированы в Mantis и ждут своего исправления.

Заключение

В работе представлена конфигурируемая ТС, которая решает следующие основные задачи:

- управление распределенной и разнородной инфраструктурой тестового стенда;
- распределение процессорных модулей целевых ЭВМ тестового стенда между задачами тестирования с учетом необходимости обеспечивать оперативную передачу этих модулей в пользование разработчиков ОСРВ;
- автоматизацию рутинных операций, связанных с подготовкой тестов к выполнению, загрузкой образов ОСРВ в процессорные модули целевых ЭВМ, сбором журналов тестирования;
- предоставление удобного интерфейса для настройки системы, постановки задач тестирования, мониторинга состояния тестового стенда, отладки и исправления обнаруживаемых проблем.

Многолетний опыт использования ТС для ОСРВ в НИИСИ РАН показал эффективность ее применения для повышения качества программных продуктов, сокращения временных затрат, ускорения процесса разработки новых версий ПО, а также упрощения процесса отладки, нахождения и исправления ошибок разработчиками ПО.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

Список литературы

1. **Система** автоматизации тестирования Avacado Framework. URL: https://avocado-framework.github.io/

2. **Система** управления тестированием операционных систем LAVA. URL: https://www.lavasoftware.org/

3. **Система** тестирования Linux Test Project. URL: http://linux-test-project.github.io/

4. **Герлиц Е. А., Кулямин В. В., Максимов А. В., Петренко А. К., Хорошилов А. В., Цыварев А. В.** Тестирование операционных систем // Труды Института системного программирования РАН. 2014. Т. 26, № 1. С. 73—108.

5. **Тестовый** набор Open POSIX Test Suite. URL: http://posixtest.sourceforge.net/

6. **Система** тестирования UnixBench. URL: https://github.com/kdlucas/byte-unixbench

7. **Система** непрерывной интеграции программного обеспечения BuildBot. URL: https://buildbot.net/

8. **Leszko R.** Continuous Delivery with Docker and Jenkins. Delivering software at scale. Packt Publishing, 2017. 332 p.

9. **van Baarsen J.** GitLab Cookbook, Packt Publishing, 2014. 172 p.

10. **Годунов А. Н., Солдатов В. А.** Операционные системы семейства Багет (сходства, отличия и перспективы) // Программирование. 2014. № 5. С. 68—76.

11. **Аряшев С. И., Зубковский П. С.** 64-разрядные системы на кристалле с архитектурой "КОМДИВ" // Наноиндустрия. 2019. № 89. С. 78—79.

12. **Система** управления сообщениями об ошибках Mantis. URL: https://www.mantisbt.org/

Configurable Test System for RTOS of BAGET Family

A. N. Godunov, nkag@niisi.ras.ru, **I. I. Khomenkov**, nkigor@niisi.ras.ru,
V. G. Shchepkov, nkvs@niisi.ras.ru, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (SRISA RAS), Moscow, 117218, Russian Federation,
A. V. Khoroshilov, khoroshilov@ispras.ru, Ivannikov Institute for System Programming of the Russian Academy of Sciences, Moscow, 109004, Russian Federation

Corresponding author:

Shchepkov Vladimir G., Leading Engineer, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (SRISA RAS), Moscow, 117218, Russian Federation
E-mail: nkvs@niisi.ras.ru

Received on January 20, 2022
Accepted on February 24, 2022

The article describes a test system designed for verification of the real-time operating system BAGET Family for embedded systems, developed and used at the Scientific Research Institute for System Analysis of the Russian Academy of Sciences (SRISA RAS). This Unix-like operating system is based on the POSIX and ARINC-653 programming standards. Of course, there is specialized software for automating testing of Unix-like operating systems, for example, Avocado, LAVA, Linux Test Project, Linux Distribution Checker, Open POSIX Test Suite, UnixBench, etc. But the use of such ready-made software systems is not always convenient since they either contain only highly specialized test suites, support only specific hardware, or do not contain a flexible configuration system. Therefore, In SRISA RAS, its own original test system was developed. The task was to create a convenient testing tool for both software testers and programmers. Many years of experience in utilizing the test system has shown the effectiveness of its use to improve the quality of software products, reduction of time spent on testing and analysis of results, maximum automation of software testing process, speed up the process of developing new software versions, as well as simplify the process of debugging, finding and fixing errors by software developers.

Keywords: *real-time operating system, RTOS, software testing, automation of program developing, cross-platform software developing, program language C, C++, POSIX, ARINC-653*

For citation:

Godunov A. N., **Khomenkov I. I.**, **Shchepkov V. G.**, **Khoroshilov A. V.** Configurable Test System for RTOS of BAGET Family, *Programmnaya Ingeneria*, 2022, vol. 13, no. 4. 168—177.

DOI: 10.17587/prin.13.168-177

References

1. **Automated** Testing Framework — Avocado, available at: https://avocado-framework.github.io/

2. **Linaro** Automated Validation Architecture — LAVA, available at: https://www.lavasoftware.org/

3. **Linux** Test Project, available at: http://linux-test-project.github.io/

4. **Gerlits E. A., Kuliamin V. V., Maksimov A. V., Petrenko A. K., Khoroshilov A. V., Tsyvarev A. V.** Testing of Operating Systems, *Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS),* 2014, vol. 26, no. 1, pp. 73—108 (in Russian).

5. **Open** POSIX Test Suite, available at: http://posixtest.sourceforge.net/

6. **UNIX** benchmark suite — UnixBench, available at: https://github.com/kdlucas/byte-unixbench

7. **Automated** Build, Test, and Release Framework — BuildBot, available at: https://buildbot.net/

8. **Leszko R.** *Continuous Delivery with Docker and Jenkins. Delivering software at scale*, Packt Publishing, 2017, 332 p.

9. **van Baarsen J.** *GitLab Cookbook,* Packt Publishing, 2014, 172 p.

10. **Godunov A. N., Soldatov V. A.** OSRT BAGET Family, *Programming,* 2014, no. 5, pp. 68—76 (in Russian).

11. **Aryashev S. I., Zubkovsky P. S.** 64-bit SOC with architecture "COMDIV", *Nanoindustry,* Moscow, 2019, vol. 89, pp. 78—79 (in Russian).

12. **Mantis** Bug Tracker, available at: https://www.mantisbt.org/

УДК 621 DOI: 10.17587/prin.13.178-186

Г. Б. Евгенев, д-р техн. наук, проф., g.evgenev@mail.ru, МГТУ им. Н. Э. Баумана

Российская технология индустрии 5.0. Метаонтология

Описаны методы и средства создания интегрированных интеллектуальных систем конструкторско-технологического проектирования в машиностроении. Введено понятие представления модулей знаний для непрограммистов.

Ключевые слова: *Индустрия 4.0, Индустрия 5.0, цифровые производства, Интернет знаний, Интернет вещей, интегрированные системы, интеллектуальные системы*

Введение

Наиболее серьезный вызов, с которым сталкивается экономика России, — это низкая эффективность. Без модернизации нельзя создать надежный фундамент повышения уровня жизни и обеспечить надежную безопасность страны в нестабильном мире. В связи с этим в состав Национальных проектов России на период с 2019 по 2024 гг. включен проект "Цифровая экономика". В числе целей этого проекта — использование отечественного программного обеспечения в области цифровых производств, а также создание сквозных цифровых технологий преимущественно на основе отечественных разработок. Ставится задача преобразования промышленности посредством внедрения цифровых технологий и платформенных решений.

Модули знаний

В настоящее время происходит четвертая промышленная революция (4ПР), для обозначения которой используется термин Индустрия 4.0. [1—4]. В работе [5] приводится интервью с представителем японской компании о развитии цифрового общества. Отмечено, что Индустрия 4.0 — лишь очередной этап в процессе с перехода к Обществу 5.0, которое представляет собой ступень, следующую за информационным обществом [6]. Общество 5.0 представляет собой оптимизацию ресурсов социума в целом через интеграцию физического пространства и киберпространства. На основе фундамента, создаваемого технологией Индустрия 4.0, в настоящее время необходимо обеспечить переход к технологии Индустрия 5.0 [7].

Настоящая статья посвящена описанию производственных методов, обеспечивающих повышение производительности труда. Статья учитывает основные положения стандарта [8]. Описываемые положения статьи моделируют жизненный цикл машиностроительного изделия и могут быть применены для всех соответствующих предприятий. Статья посвящена информационному классу систем и основана на модульном принципе, разработанном автором.

Поскольку здесь речь идет об интеллектуальных системах, то важнейшим элементом является модуль знаний (МЗ), в качестве которого выбран модуль стандарта IDEF0 (рис. 1), представляемый на языке деловой прозы.

Особенностью стандарта IDEF0 является акцент на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между модулями. Описание МЗ на языке деловой прозы выглядит как "черный ящик" с входами, выходами, управлением и механизмом, который постепенно детализируется до необходимого уровня. В IDEF0 существуют словари описания активностей и стрелок, в этих словарях можно дать описания того, какой смысл вкладывается в ту или иную активность либо стрелку.

Описание методологии IDEF0 содержится в рекомендациях Р 50.1.028-2001 "Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования".

Отображаются все сигналы управления. Модель в рамках IDEF0 используется при организации бизнес-процессов и проектов, основанных на моделировании всех процессов.

Формирование МЗ осуществляется с помощью системы СПРУТ-ЭксПро [9]. В модуле на рис. 2 проводится расчет по формулам значений ряда величин.

С помощью МЗ для описания формул можно формировать текстовые переменные, например, обозначения изделий, тексты содержания технологических операций, переходов и т. д. На рис. 3 приведен пример формирования содержания перехода механической обработки в соответствии со стандартом ЕСТД (Единой системой технологической документации). При значениях входных переменных Per = "Точить",

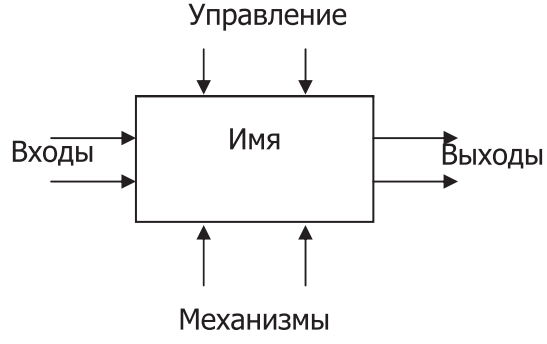

Рис. 1. Внешнее представление модуля в стандарте IDEF0

Модуль: V13
Разработчик: Евгенев Г. Б.
Наименование: Расчет номинальной величины деформации
Источник информации: Шувалов С. А. Методические указания по расчету волновых зубчатых передач на ЭВМ. Изд. МГТУ им. Н.Э. Баумана, 1987

Наименование	Имя	Ограничение
Тип редуктора	ТипРед	волновой одновенцовый
Передаточное отношение заданное	uz	(0,)
Число зубьев гибк. Колеса предвар.	zf	
Коэф. Увеличения вращ. Момента при пуске	K1	1.9
Номинальная вел. Радиальной деформации	NWo	0.84+0.001*uz+1.6*10^(-3) *K1*uz^(1/2)+0.15*10^(-3) *K1*uz
Глубина захода зубьев допуст., мм	hd	4*NWo-(4.6-4*NWo)*zf/10^3-2.45

Рис. 2. Внешнее представление комбинированного модуля

Модуль: TKP3
Разработчик: Евгенев Г. Б.
Наименование: Формирование содержания перехода
Источник информации: ЕСТД

Наименование	Имя	Ограничение
Переход обработки резанием	Per	
Элемент обрабатываемый	ElObr	
Номер элемента	NoEl	[1,)
Дополнит. информация перехода 2	DinPer2	
Дополнит. информация перехода 4	DinPer4	
Количество элементов	KolEl	
Номер элемента строковый	NoElStr	STR(NoEl:0)
Содержание перехода	SodPer	Per+" "+ DinPer2+" "+ ElObr+" "+ NoElStr+" "+ DinPer4

Рис. 3. Внешнее представление модуля — формулы формирования текстовой переменной

ElObr = "канавку", NoEl = 1, DinPer2 = "кольц.", DinPer4 = "окончательно" содержание перехода будет иметь такой вид: "Точить кольц. канавку 1 окончательно". Функция STR обеспечивает перевод данных из числовой формы в строковую.

Функциональные зависимости часто имеют табличную форму представления. Для ввода таких зависимостей в базы знаний используются МЗ с механизмами в виде таблиц (рис. 4).

В системе, построенной на основе программных средств СПРУТ [11], геометрические и сложные математические вычисления не могут быть представлены в форме МЗ. Для использования математических знаний введены модули с механизмами в виде программных модулей. Пример такого модуля приведен на рис. 5. Этот МЗ предназначен для генерации чертежа спроектированной детали. Аналогичным образом могут генерироваться поверхностные и твердотельные модели изделий, а также обращение к программным средствам, созданным вне среды СПРУТ.

Сгенерированные из элементарных МЗ методы (сложные функции) могут быть представлены в виде МЗ и использованы для решения комплексных задач.

В системе СПРУТ предусмотрена возможность организации циклических процессов. Циклы генерируются автоматически при появлении в выходных переменных одного из модулей метода, выполняющего функцию управления повторением цикла выделенной переменной с идентификатором FinCalc. В методе может содержаться только один цикл. Для установки переменных цикла используются модули инженерных знаний без входных переменных, которые не включаются в тело цикла, располагаясь перед ним.

Цифровая революция должна дать возможность непрограммирующему носителю знаний вводить их в компьютер без посредников. Это стало возможным благодаря методологии *экспертного программирования* [12]. В этой методологии знания, как описано выше, излагаются на языке *деловой прозы*. Этот язык максимально приближен к литературному языку, но

М3: "NzTpzbCh" - Назначение Тпз базового для червячных колес

Предусловия запуска

имя	наименование	тип	условие
ViZubKol$	Вид зубчатого колеса	STRING	червячное

Входные свойства

имя	наименование	тип	значение
ZamPris$	Замена приспособлений	STRING	
VidPod$	Вид подачи червячной модульной фрезы	STRING	
HarNal$	Характеристика наладки	STRING	
m_	Модуль детали, мм	REAL	

Механизм - Таблица

Конфигурация свойств в таблице

		ZamPris$
		m_
HarNal$	VidPod$	tpzb

Таблица

		с заменой установочных приспособлений			без замены установочных приспособлений		
		(0,6]	(6,12]	(12,)	(0,6]	(6,12]	(12,)
без замены фрезерного суппорта	радиальная	29	38	47	17	23	27
	тангенциальная	31	40	50	19	24	30
с заменой фрезерного суппорта	радиальная	39	52	67	27	37	42
	тангенциальная	41	56	72	29	40	52

Выходные свойства

имя	наименование	тип	значение
tpzb	Норматив подготовительно-заключительного времени базовый, мин	REAL	

Рис. 4. Внешнее представление модуля — таблицы

```
Модуль: M8
Разработчик: Евгенев Г. Б.
Наименование: формирование чертежа
Источник информации: Анурьев В.И. Справочник конструктора, т.2, стр.7
```

Наименование	Имя	Ограничение
Тип оси	TO	ось гладкая
Диаметр оси стандартный, мм	D	(0 , 50]
Длина оси стандартная, мм	L	
Ширина фаски, мм	c	
Чертеж детали	AXLE	AXLES.prt

Рис. 5. Внешнее представление модуля — геометрической процедуры: имя — наименование сегмента графической базы; ограничение — имя программы AXLES.prt

формализован настолько, что имеется возможность автоматической генерации программных средств, соответствующих исходным текстам.

Интеллектуальное производство

Интеллектуальное производство — продукт человеческого интеллекта, деятельность по созданию интеллектуальных и материальных благ. Потребление этих ценностей обеспечивает личное и общественное благосостояние. Интеллектуальный капитал выступает как основной воспроизводимый фактор производства.

Интеллектуальный капитал (ИК) — это знания, навыки и производственный опыт конкретных людей и нематериальные активы. Знания эти включают патенты, базы данных, программное обеспечение, товарные знаки и др., которые используются в це-

лях максимизации прибыли и других экономических и технических результатов.

Разные сочетания способов увеличения производительных сил экономической системы определяют ее структуру и динамику развития. По определению К. Маркса, "Экономические эпохи различаются не тем, что производится, а тем, как производится, какими средствами труда" [10]. В связи с этим значимость отдельных видов ресурсов изменяется по мере перехода от доиндустриальной к индустриальной, а от нее к постиндустриальной технологии.

В доиндустриальном обществе приоритет принадлежал природным и трудовым ресурсам, в индустриальном обществе — материальным ресурсом, в постиндустриальном обществе — интеллектуальным и информационным ресурсам. В настоящее время технологическая революция с информационными технологиями в центре заново формирует материальную основу общества. В новой информационной экономике — экономике, основанной на знаниях, — источник производительности заключается в технологии генерирования знаний.

Понятие "информационная экономика" (как и информационное общество) было введено в научный оборот в начале 1960-х гг. Знания и информация являются критически важными элементами во всех экономических системах, так как процесс производства всегда основан на некотором уровне знаний и на обработке информации.

Согласно определению К. Маркса [10] "Развитие основного капитала является показателем того, до какой степени всеобщее общественное знание превращается *в непосредственную производительную силу*, и отсюда — показателем того, до какой степени условия самого общественного жизненного процесса подчинены контролю всеобщего интеллекта и преобразованы в соответствии с ним". Современное изменение технологической парадигмы рассматривают как сдвиг от технологии, основанной главным образом на вложении дешевой энергии, к технологии, основанной преимущественно на дешевых вложениях знания и информации, ставших предметом и средством труда. Впервые в истории человеческая мысль прямо является производительной силой, а не просто определенным элементом производственной системы. Характеризуя условия формирования массового производства, К. Маркс отмечал [10]: "впервые в крупных масштабах подчиняет непосредственному процессу производства *силы природы* ... Эти силы природы как таковые *ничего не стоят*". В условиях новой постиндустриальной экономики изменились не виды деятельности человечества, а технологическая способность использовать в качестве прямой производительной силы то, что отличает человека от других биологических созданий, а именно — способность обрабатывать и понимать символы.

Вместе с тем в этих новых экономических условиях особую актуальность приобретает положение К. Маркса [10] о важности индивидуальных знаний в применении науки для анализа процесса производства (традиционных сведений, наблюдений, профессиональных секретов, полученных экспериментальным путем). Такая актуальность обусловлена применением естественных наук к материальному производству.

Концептуальная модель интеллектуальных производств

Существующая теория производственной фирмы [11—13] основана на детерминированной модели, в то время как среда окружения для большинства достаточно крупных фирм является стохастической. В соответствии с законом разнообразия такая модель может быть полезной только для анализа отдельных статических ситуаций. При этом компания не способна выжить в реальных условиях рыночной экономики. Онтологическая модель экономики, основанной на знаниях, сводит ее в плоскость инновационной экономики с единственной формой приращения стоимости (получения ценностей) в виде новых технических результатов.

Отсюда следует, что само производство должно относиться к категории гибких производственных систем. В соответствии со стандартом гибкая производственная система — это управляемая средствами вычислительной техники совокупность технологического оборудования, состоящего из разных сочетаний гибких производственных модулей и (или) гибких производственных ячеек. В такой системе должны присутствовать автоматизированная система технологической подготовки производства и система обеспечения функционирования. Система подготовки производства должна обладать свойством автоматизированной переналадки при изменении программы производства изделий, разновидности которых ограничены технологическими возможностями оборудования. Такая формулировка дает постиндустриальную характеристику гибкой производственной системы, схема которой изображена далее на рис. 6. На этом рисунке приведена схема метаонтологии интеллектуальных производств. Метаонтология содержит общие понятия и отношения, независящие от предметной области. Отсюда следует, что само производство должно относиться к категории гибких производственных систем.

На рис. 6 описание производства дается с точки зрения как предметной онтологии, так и онтологии знаний. С предметной точки зрения гибкая производственная система — это управляемая средствами вычислительной техники совокупность технологического оборудования. Эта система состоит из разных сочетаний гибких производственных модулей и (или) гибких производственных ячеек.

С точки зрения знаний в состав гибкой производственной системы входят автоматизированная подсистема технологической подготовки производства и подсистема обеспечения функционирования. Подсистема поддержки функционирования обладает свойством автоматизированной переналадки при изменении программы производства изделий, разновидности которых ограничены технологическими возможностями оборудования.

При переходе к постиндустриальным технологиям сюда должны быть включены все компоненты

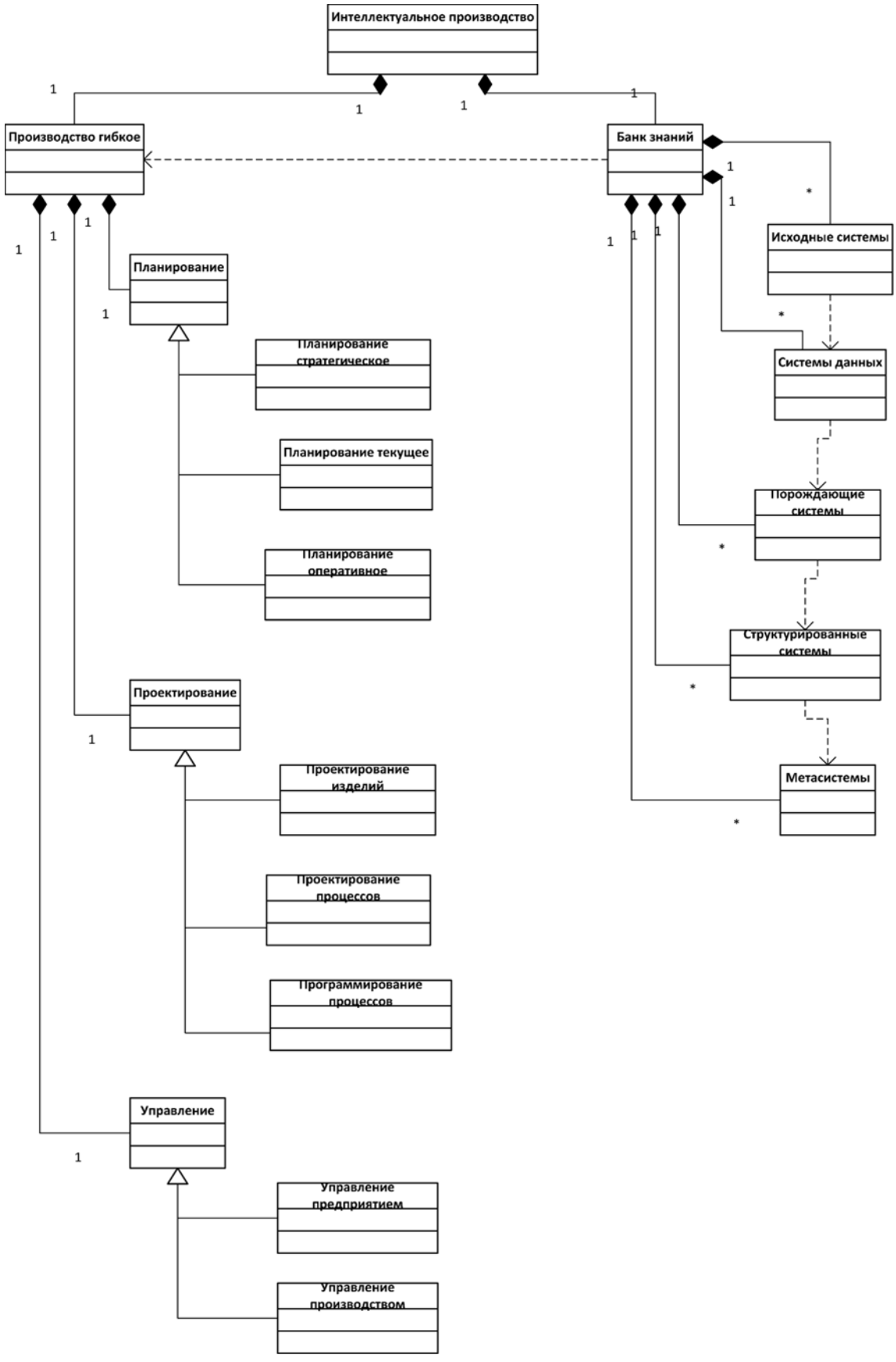

Рис. 6. Концептуальная модель интеллектуальных производств

жизненного цикла предприятий и изделий. В этот цикл входят: планирование, включая стратегическое, текущее и оперативное; проектирование, включая проектирование изделий и процессов, программирование обработки; управление предприятием и производством.

Банк знаний состоит из следующих взаимосвязанных компонентов: исходные системы; системы данных; порождающие системы; структурированные данные; метасистемы. Эти компоненты описывают различные *эпистемологические уровни,* т. е. уровни знания относительно рассматриваемых явлений.

Настоящая статья представляет собой изложение материалов в соответствии представленной на рис. 6 метаонтологией, которая определяет ее структуру (см. таблицу).

Любая проектируемая *система* состоит из элементов и связей между ними [14]. Формально структуру системы (изделия или процесса) можно представить в виде упорядоченной пары $S = < A, R >$, где A — множество элементов системы; R — множество отношений между этими элементами.

Отсюда следует, что классификация проектируемых систем может быть построена с использованием одного из двух фундаментальных критериев различия: *а* — по типу элементов, образующих систему; *б* — по типу отношений, связывающих эти элементы в систему [14]. Эти классификационные критерии можно рассматривать как независимые.

Примером использования критерия *а* служит традиционное разделение науки и техники на дисциплины и специальности, каждая из которых изучает элементы определенных типов. Применительно к конструированию изделий — это разделение в соответствии с классификацией машин и аппаратов, например, по системе ЕСКД (Единая система конструкторской документации). Применительно к технологии — это классификация по технологическим методам в соответствии с ЕСТД (например, литье, обработка давлением, резанием, термическая и т. п.). Поскольку элементы разных типов требуют разных экспериментальных средств для сбора данных, эта классификация имеет экспериментальную основу.

Критерий *б* дает совершенно иную классификацию систем, а именно — класс определяется типом отношений, а тип элементов, на которых заданы эти отношения, не фиксируется. Такая классификация связана с обработкой данных, а не с их сбором, и основа ее преимущественно теоретическая. До появления средств вычислительной техники в качестве критерия *б* могли использоваться лишь типы математических моделей проектируемых систем, так как математические модели имеют аксиоматическое построение и абстрагированы от материальной и энергетической сущностей явлений.

Математические модели одного и того же класса могут использоваться для описания связей свойств элементов различной материальной сущности. Вместе с тем во второй половине XX века появился ряд новых родственных теоретических дисциплин, непосредственно связанных с развитием вычислительной техники. К их числу относятся: кибернетика;

Традиционная классификация прикладных областей и эпистемологическая классификация уровней знаний

Традиционная классификация прикладных областей									Эпистемологическая классификация
Наука			Техника			Другие области			Эпистемологические уровни
Физика	Химия	..	Механика	Электроника	..	Медицина	Музыка	..	
		..			..			..	Уровни 4, 5 МЕТАСИСТЕМЫ (отношения между определенными ниже отношениями)
		..			..			..	Уровень 3 СТРУКТУРИРОВАННЫЕ СИСТЕМЫ (отношения между определенными ниже системами)
		..			..			...	Уровень 2 ПОРОЖДАЮЩИЕ СИСТЕМЫ (модели, генерирующие определенные ниже данные)
		..			..			...	Уровень 1 СИСТЕМЫ ДАННЫХ (данные, структура которых определена ниже)
		..			..			...	Уровень 0 ИСХОДНЫЕ СИСТЕМЫ (язык определения данных)

общесистемные исследования; математическая теория систем; теория принятия решений; исследование операций; теория искусственного интеллекта.

Эти дисциплины обладают одним общим свойством — они связаны с такими системными задачами, в которых главенствующими являются не тип сущностей, из которых состоит система, а информационные, реляционные и структурные аспекты.

Самыми большими по критерию *б* являются классы, описывающие различные *эпистемологические уровни,* т. е. уровни знания относительно рассматриваемых явлений. Далее они уточняются с помощью различных методологических отличий. Каждый выявленный таким образом класс подразделяется далее на меньшие, состоящие из систем, эквивалентных с точки зрения конкретных, практически существенных сторон, определенных в них отношений. Поскольку системы в каждом из таких классов эквивалентны только с точки зрения некоторых характеристик их отношений, они могут базироваться на совершенно разных типах элементов. Если рассматривать только характеристики отношений в системах, то достаточно каждый такой класс систем заменить одной системой, представляющей этот класс.

В область *системологии* [14] входят все типы свойств отношений, существенные для отдельных классов или для всех систем. Выбранная классификация систем по отношениям определяет способ разбиения системологии на подобласти, так же как традиционная наука подразделяется на различные дисциплины и специальности. Знания в науке о системах, т. е. знания, относящиеся к различным классам свойств отношений в системах, можно получать либо с помощью математики, либо с помощью экспериментов с их моделями на ЭВМ.

Системная методология представляет собой совокупность методов изучения свойств различных классов систем и решения системных задач, т. е. задач, постановки которых касаются отношений в системах. Ядром системологии является классификация систем с точки зрения таких отношений. Главная задача системной методологии — предоставление в распоряжение потенциальных пользователей из разных дисциплин и предметных областей методов решения всех типов характерных им задач.

Основой иерархической классификации систем в системологии является иерархия их эпистемологических уровней (см. таблицу). Эта иерархия опирается на следующие элементарные понятия:

- исследователь (конструктор, технолог) и его среда;
- исследуемая (проектируемая) система и ее среда;
- взаимодействие между исследователем и системой.

Самый нижний уровень в этой иерархии обозначается как нулевой (0) уровень. На этом уровне система определяется через множество свойств (переменных), множество потенциальных состояний (значений) этих свойств и операционный способ описания смысла этих состояний в терминах значений соответствующих атрибутов данной системы. Для систем, определенных на этом уровне, используется термин *исходная система,* указывающий на то, что она является, по крайней мере, потенциально, источником эмпирических данных. В литературе используется также название "система без данных", означающее, что система этого уровня представляет собой простейшую стадию процесса исследования, не использующую данные о доступных переменных.

Иными словами, на уровне 0 рассматриваются характеристики и взаимосвязи между свойствами (переменными) исследуемой (проектируемой) системы. В качестве теоретической основы на этом уровне может использоваться математическая лингвистика, изучающая понятия, имена, термины и связи между ними. Для специалистов в области автоматизации проектирования язык выступает как средство описания исходных систем. Лексические единицы языка, используемые для обозначения исходных систем, называются *понятиями*. Понятие обозначает не какой-то конкретный предмет, а класс однородных предметов (процессов). Понятиям соответствует некоторая *структура,* определяющая набор обязательных характеристик класса предметов. Характеристики в языке обозначаются *именами,* а конкретные значения имен — *терминами*. С помощью терминов из класса предметов выделяется один конкретный.

На более высоких эпистемологических уровнях системы отличаются друг от друга содержанием знаний относительно переменных соответствующей исходной системы. На более высоком уровне используются все знания расположенных ниже по системной иерархии систем и, кроме того, дополнительные знания, недоступные низшим уровням. После того как исходная система дополнена данными, т. е. фактическими состояниями основных переменных при определенном наборе параметров, рассматривают новую (исходную систему с данными), как определенную на эпистемологическом уровне 1. Системы этого уровня называются *системами данных*. В зависимости от задачи данные могут быть получены из наблюдений или с помощью измерений (как в задаче моделирования) или определены как желательные состояния (в задаче проектирования).

При конструировании используются данные архива ранее спроектированных конструкций, данные об имеющихся унифицированных сборочных единицах и деталях, о типовых конструкторско-технологических элементах деталей и т. п. При проектировании технологических процессов используются данные архива ранее спроектированных процессов, данные об имеющихся унифицированных технологических процессах, средствах технологического оснащения и т. п.

Более высокие эпистемологические уровни содержат знания об отношениях рассматриваемых переменных, посредством которых можно генерировать данные при соответствующих начальных и граничных условиях. Эти характеристики отношений содержатся обычно в нормативной технической документации, иногда их источником является практический инженерный опыт.

Уровень 2 применительно к задачам автоматизации проектирования представляет собой уровень

базы знаний генерации значений переменных, определяющих свойства изделий и технологических процессов. Здесь задаются инвариантные параметрам функциональные связи основных переменных, в число которых входят определяемые соответствующей исходной системой переменные и, возможно, некоторые дополнительные. Каждая дополнительная переменная определяется конкретным правилом преобразования на множестве параметров, применимом или к основной переменной исходной системы, или к гипотетической (ненаблюдаемой), введенной пользователем (составителем модели). Эту переменную называют *внутренней*. Каждое правило преобразования базы знаний на этом уровне обычно представляет собой однозначную функцию, присваивающую каждому элементу множества переменных, рассматриваемому в этом правиле в качестве выходного, единственное значение из множества допустимых.

Поскольку задачей генерации свойств является реализация процесса, при котором состояния основных переменных могут порождаться по множеству параметров при любых начальных или граничных условиях, системы уровня 2 называются *порождающими* системами *(generative system)*. При конструировании на уровне 2 располагаются базы знаний, связанные с расчетом конструкций, при проектировании технологических процессов — базы знаний по выбору заготовок, формированию набора переходов, расчету режимов обработки, норм времени и т. п.

На эпистемологических уровнях 4 и выше системы состоят из набора определенных на более низком уровне систем и некоторой *метахарактеристики* (правила, отношения, процедуры), описывающей изменения в системах более низкого уровня. Такие системы называются *метасистемами*. Требуется, чтобы метасистемы имели одну и ту же исходную систему и были определены на уровне 1, 2 или 3.

Заключение

Представлены базовые положения и методы создания интеллектуальных производств в рамках концепции "Индустрия 5.0". Такие положения рассматриваются как подходы на развитие технологий Интернета вещей и принципов Интернета знаний. Описана концептуальная модель таких производств. Дано описание банка знаний и эпистемологических уровней представления знаний.

Список литературы

1. **Industry 4.0:** the fourth industrial revolution — guide to Industrie 4.0. URL: https://www.i-scoop.eu/industry-4-0.
2. **Industry 4.0:** the Future of Smart Manufacturing — Praim. URL: https://www.praim.com/en/news/industry-4-0-the-future-of-smart-manufacturing/
3. **Siemens** | Industrie 4.0. URL: https://www.siemens.com/digital/enterprise.
4. **Digital** Factory 4.0 | Industry 4.0 solution. URL: https://antsolutions.eu/
5. **Норицугу У.** Общество 5.0: взгляд Mitsubishi Electric // Экономические стратегии. 2017. № 4. С. 2—11.
6. **Ржевский Г. А.** Самоорганизация в социальных системах // Онтология проектирования. 2014. № 4 (14). С. 8—17.
7. **Концепция** Индустрии 5.0. URL: http://industry5.ru/concept
8. **ГОСТ Р 1.0-2004** "Стандартизация в Российской Федерации. Основные положения".
9. **Евгенев Г. Б.** Основы автоматизации технологических процессов и производств. Т. 1: Информационные модели. Т.2: Методы проектирования и управления. М.: Изд-во МГТУ им. Н. Э. Баумана, 2015. Т. 1: 441 с. Т. 2: 479 с.
10. **Маркс Карл** // Большая советская энциклопедия: [в 30 т.] / под ред. А. М. Прохорова. 3-е изд. М.: Советская энциклопедия, 1969.
11. **СПРУТ**-Технология. Автоматизация проектирования. URL: https://sprut.ru/
12. **Евгенев Г. Б.** Экспертопедия как средство создания онтологического интернета знаний // Онтология проектирования. 2019. Т. 9, № 3 (33). С. 307—320.
13. **Центр** СПРУТ. Высокопрофессиональные российские программные решения для предприятия. URL: https://csprut.ru/
14. **Клир Дж.** Системология. Автоматизация решения системных задач: Пер. с англ. М.: Радио и связь, 1990. 544 с.

Russian Technology of Industry 5.0. Metaontology

G. B. Evgenev, g.evgenev@mail.ru, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation

Corresponding author:

Evgenev Georgiy B., Professor, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation
E-mail g.evgenev@mail.ru

Received on January 20, 2022
Accepted on February 24, 2022

The methods and means of creating integrated intelligent systems for design and technological design in mechanical engineering are described. Introduced representation of knowledge modules for non-programmer

***Keywords*:** *Industry 4.0, Industry 5.0, digital production, Internet of knowledge, Internet of things, integrated systems, intelligent systems*

For citation:

Evgenev G. B. Russian Technology of Industry 5.0. Metaontology, *Programmnaya Ingeneria*, 2022, vol. 13, no. 4, pp. 178—186.

DOI: 10.17587/prin.13.178-186

References

1. **Industry 4.0:** the fourth industrial revolution — guide to Industrie 4.0, available at: https://www.i-scoop.eu/industry-4-0.

2. **Industry 4.0:** the Future of Smart Manufacturing — Praim, available at: https://www.praim.com/en/news/industry-4-0-the-future-of-smart-manufacturing/

3. **Siemens** | Industrie 4.0, available at: https://www.siemens.com/digital/enterprise

4. **Digital** Factory 4.0 | Industry 4.0 solution, available at: https://antsolutions.eu/

5. **Noritsugu U.** Society 5.0: a view of Mitsubishi Electric, *Economic strategies*, 2017, no. 4, pp. 2—11.

6. **Rzhevsky G. A.** Self-organization in social systemg, *Ontolodia proektirovaniya,* 2014, no. 4 (14), pp. 8—17 (in Russian).

7. **The concept** of Industry 5.0, available at: http://industry5.ru/koncept (in Russian).

8. **GOST** R 1.0-2004 "Standardization in the Russian Federation. Basic provisions" (in Russian).

9. **Evgenev G. B.** Basics of automation of technological processes and production. Vol. 1: Information models. Vol. 2: Methods of design and management, Moscow, Publishing house of MSTU im. N. E. Bauman, 2015, vol. 1: 441 p. vol. 2: 479 p. (in Russian).

10. **Karl Marx**, *Great Soviet Encyclopedia:* [in 30 volumes] / ed. A. M. Prokhorova, 3rd ed, Moscow, Soviet Encyclopedia, 1969 (in Russian).

11. **SPRUT-**Technology. Design Automation, available at: https://sprut.ru/ (in Russian).

12. **Evgenev G. B.** Experttopedia as a means of creating an ontological Internet of knowledge, *Ontolodia proektirovaniya,* 2019, vol. 9, no. 3 (33), pp. 307—320 (in Russian).

13. **SPRUT** Center. Highly professional Russian software solutions for the enterprise, available at: https://csprut.ru/ (in Russian).

14. **Clear J.** Systemology. *Automation of solving system problems*: Per. from English, Moscow, Radio and communication, 1990, 544 p. (in Russian).

ИНФОРМАЦИЯ

DOI: 10.17587/prin.13.187-199

Applying Unsupervised Machine Learning Algorithms to Ensure Requirements Consistency[1]

K. I. Gaydamaka, k.gaydamaka@gmail.com, Department of Systems Engineering, "MIREA — Russian Technological University", Moscow, 119454, Russian Federation,
A. D. Belonogova, alena.belonogova@yandex.ru, National Research Nuclear University "MEPhI", Moscow, 115409, Russian Federation

Corresponding author:
Gaydamaka Kirill I, Postgraduate Student, Department of Systems Engineering, "MIREA — Russian Technological University", Moscow, 119454, Russian Federation
E-mail: k.gaydamaka@gmail.com

Received on November 16, 2021
Accepted on March 05, 2022

The article is devoted to the problem of ensuring the quality of requirements for complex technical systems. The purpose of this article is to apply unsupervised machine learning techniques to test a set of requirements for consistency. It is assumed that the clustering of requirements will allow us to determine the requirements that are closest in meaning to the given one, and this may indicate a possible contradiction and require additional check of potentially conflicting requirements. The article discusses a comparison of clustering methods such as k-means, agglomerative hierarchical clustering, DBSCAN, EM-algorithm, as well as methods for converting sentences into numeric vectors TF-IDF, doc2vec, BERT. BERT and k-means showed the best result — a cluster that included only conflicting requirements.

Keywords: *natural language processing, requirements clustering, requirements management, requirements engineering, requirements consistency*

For citation:

Gaydamaka K. I., **Belonogova A. D.** Applying Unsupervised Machine Learning Algorithms to Ensure Requirements Consistency, *Programmnaya Ingeneria*, 2022, vol. 13, no. 4, pp. 187—199.

DOI: 10.17587/prin.13.187-199

УДК 004.852

К. И. Гайдамака, аспирант, k.gaydamaka@gmail.com, РТУ МИРЭА,
А. Д. Белоногова, магистр, alena.belonogova@yandex.ru,
Национальный исследовательский ядерный университет "МИФИ"

Применение методов машинного обучения без учителя для обеспечения непротиворечивости требований

Статья посвящена проблеме обеспечения качества требований к сложным техническим системам. Цель этой статьи — применить методы машинного обучения без учителя, чтобы проверить набор требований на непротиворечивость. Предполагается, что кластеризация требований позволит определить наиболее близкие по смыслу к заданному требования, а это может указывать на возможное противоречие и потребовать дополнительной проверки потенциально конфликтующих требований. Рассматривается сравнение таких методов кластеризации, как k-means, агломеративная иерархическая кластеризация, DBSCAN, EM-алгоритм, а также методы преобразования предложений в числовые векторы TF-IDF, doc2vec, BERT. Наилучшие результаты показал BERT в тандеме с k-means, что позволило получить кластер, в который включены только конфликтующие требования.

Ключевые слова: *обработка естественного языка, обучение без учителя, кластеризация, управление требованиями, инженерия требований, непротиворечивость требований*

[1] The article is based on the materials of the report at the Seventh International Conference "Actual problems of Systems and Software Engineering" APSSE 2021.

Introduction

Requirements make it possible to determine what stakeholders want to get from the system being created and what properties the system should have to satisfy their needs [1]. They are necessary so that the project team has a clear and consistent idea of what system will need to be developed, what functionality this system should have and what problem the end user should solve. One of the key and important stages of work on a project is the stage of developing system requirements. It is at this stage that it becomes possible to designate the boundaries of the problem area, then to select from it a specific list of problems, and when formulating requirements, compare each problem to some specific solution. The result of such work will be a description of the concept of a system that will be able to completely solve all the client's problems, and therefore will be economically profitable [2].

The development of high-quality system requirements plays a decisive, key role in the process of working on a project, regardless of its scale, be it the design of a nuclear power plant or the creation of a portable audio player. The timing of implementation, the cost of development and other indicators of the project's effectiveness directly depend on how accurately and correctly the system requirements describe the current reality, determine the key "problem" of the client and, in a language that is simple and understandable to all members of the project team, describe the criteria that the system must satisfy in the final the end result [3]. Cases where insufficient resources are allocated to requirements development certainly entail risks related to the earliest stages of the project work life cycle. And as you know, risks of this nature can cause the most significant damage to the cost and timing of the project. This is why it is necessary to pay special attention to requirements engineering in design practices.

Taking into account the fact, how complex large-scale projects can be, it would be logical to assert that the number of requirements in them can amount to several thousand. And since the development of requirements is entirely on the shoulders of analysts, it would be logical to assume that it cannot but depend on the human factor: inattention, negligence, haste. But in cases where huge sums of money are at stake, and due to the imperfection of the human resource, the project executor may suffer significant losses, it is necessary to minimize the human factor.

An excellent example of the situation described above would be a possible conflict of system requirements with each other. When several thousand requirements describe a large-scale system, and a staff of analysts is working on their development, it is easy to overlook such a case, as if requirement #78 said "The kettle must boil in at least 50 seconds." not less than 60 s. How easy it is to make such an annoying mistake, it is just as difficult for a person to find it, and only an automated intelligent tool can help improve the situation. This is precisely the purpose of this work — to provide an automated consistency check of system requirements.

In particular, it is assumed that the developed algorithms will make it possible to determine the requirements that are closest in semantic meaning to a certain given one, after which the algorithm will issue a recommendation to re-check the requirements from the found group for contradictions with each other. Undoubtedly, one cannot expect that the system will unambiguously and accurately determine the presence or absence of the aforementioned problem, however, one can definitely count on the fact that it will be able to competently separate the requirements by meaning and divide them into a large number, depending on the number of requirements, semantic groups — clusters. And taking into account the amended condition, the size of each cluster will not exceed two to four elements, and the presence of requirements in one cluster can potentially signal the problem of inconsistency. In this case, the analyst is left with checking several proposals, instead of a full-fledged revision of the requirements.

1. Writing the requirements

Regardless of what area the system being developed belongs to, what scale it is, the requirements should be easy to read and easy to work with.

For easy readability, it is assumed that the wording of the requirement should follow some special pattern, such that it will be convenient for the user to perceive it [4]. For example, [User Category] must have the [Capability] feature. The analyst's tasks are:

- a competent choice of a requirement template;
- substitution of values of quantities in the corresponding gaps in the requirement template.

Using patterns allows you to generate requirements at any time without spending more resources on it than possible, as well as:

- simplifies the process of changing several requirements of the same type — it is enough to change the parent template;
- simplifies the process of searching for requirements, for example, by filtering by a specific value of the initiating event;
- allow you to encapsulate confidential information, since it can be stored in a closed loop, and the process of forming a request will only have an interface for obtaining a request template, substituting secret data in the appropriate sections.

In the case of light work, the requirements should provide a range of possibilities. Below is a sample list.

- The ability to distinguish between requirements by unique identifiers.
- The ability to classify requirements for a certain set of parameters.
- Ability to monitor changes in the statuses of requirements.
- The ability to evaluate and analyze requirements as part of a whole (requirements specification).
- The ability to search the specification of requirements in order to find a specific requirement based on the context.
- The ability to establish relationships between requirements.

Based on this, it becomes obvious that it would be bad form to match the unique identifier of the requirement with the content of its content. Therefore, you should use meta information for each unit of the requirements

specification, and put some defining requirements in the meta information. Requirement attributes are called these values. The use of attributes will allow, firstly, to distinguish requirements as separate objects with unique identification numbers, which further implies the possibility of establishing links between objects. Second, the presence of meta information for each requirement allows, without reading into the text of the requirement itself, to obtain important characteristics just by simply "skimming" a glance at the specification. Third, meta information denies the overload of the requirement text — there is no need to indicate the priority, the verification method in the text itself, because such data can be taken out into meta information. And finally, the last, search capabilities are multiplied many times, if the search can be carried out not only in the text of the requirement, but also in the fields of meta information.

However, these tips are not enough to ensure the quality of the requirements. The quality of the requirements can be judged by the presence of the following properties:

- uniqueness — each requirement must have a unique identifier to distinguish it from others;
- feasibility — there must be an opportunity to implement the requirement within a specified time frame and with a specified budget;
- legality — the requirement must not contradict applicable laws;
- clarity — there should be no ambiguity in the requirement;
- accuracy — the requirement should be concise;
- verifiability — a strategy should be in place to verify that the requirement has been met;
- abstractness — the requirement should not define the implementation.

With regard to the dataset, there are the following characteristics:

- completeness — the requirements cover the entire functionality of the system;
- consistency — there are no conflicting requirements;
- no redundancy — there are no repetitions in the requirements;
- modularity — requirements that are similar in meaning are in the same category;
- structuredness — there is a clear structure in the requirements specification;
- satisfaction — all requirements marked "satisfied by" refer to existing requirements;
- testability — requirements are covered by tests at the desired level.

Inconsistency of a requirement or a set of requirements even with one criterion indicates that the specification is of poor quality, and that problems are likely to arise during the further design and implementation of the system. Therefore, the only solution is the timely correction of the error at the earliest stages of the project development. First of all, in order to make changes to the incorrect wording, it is necessary to detect the problem. However, when the requirements specification is calculated, in the case of large-scale projects with tens of thousands of requirements, the problem lies precisely in finding an error (even though it may not exist at all). In this case, an expert team is involved, which re-checks the wording of the requirements. But if the discrepancy between a separate requirement and a separate criterion is checked rather quickly (although a full check still takes a long time), then checking a set of requirements can take up a significant amount of resources, even without taking into account the human factor (i. e., that the error will not be detected even after repeated verification). Therefore, in this case, it makes sense to create an intelligent system that will help the team of systems engineers in requirements management processes.

In this work we will focus on one criterion. As part of this work, it is necessary to create a system that can read a set of requirements and check them for consistency. In other words, this system will determine the closest in meaning requirements to a user-specified requirement and issue a recommendation to check three or four proposals instead of the entire specification.

2. Description of the approach to identifying conflicting requirements

In the previous section, we identified the core of requirements engineering and requirements management discipline — the resource-intensive process of checking a set of requirements for inconsistency, including human factors. But it is necessary to assume how the invited expert would solve this problem? Of course, the easiest way is to consistently go through the list of requirements and, making pairs "each requirement with each requirement", check them for contradiction. But, obviously, this process is incredibly time consuming. Assuming that it will take ≈ 20 s to compose and verify one pair of requirements, how long will it take to verify a requirement specification consisting of 500 elements? Applying the formula (1) combinations of 2 elements out of 500 from combinatorics, we carry out elementary calculations:

$$C_n^k = \frac{n!}{k!(n-k!)} \to C_{500}^2 = \frac{500!}{2! \times 498!} = \\ = 124\,750 \text{ pairs} \to 124\,750 \cdot 20 \approx 693 \text{ hours}. \quad (1)$$

Obviously, this is not possible to spend about a month without interruption on such a check "head-on". Most likely, the requirements specialist would try to divide the specification into logical subgroups if the requirements were in the same subgroup — this means that they are similar in meaning. And then, instead of checking the consistency of the combination of all requirements with all, he would perform a similar task within one small structure. While we do not take into account the process of dividing the requirements into groups, we will assume that this task has already been completed: there are 50 groups of 10 requirements each. Then our formula for calculating the amount of time spent on the expert assessment of consistency will look like this:

$$C_n^k = \frac{n!}{k!(n-k!)} \to C_{10}^2 = \frac{10!}{2! \cdot 8!} = \\ = 45 \text{ pairs} \to (45 \times 20) \times 50 = 12.5 \text{ hours}. \quad (2)$$

Thus, the time was reduced by an order of magnitude, however, checking 50 groups of 15 minutes each, which

is still quite a long time, but at least once this task can be completed.

In addition, it is possible to radically change the approach to solving this problem — *not to check all the requirements in the aggregate, but to check the contradiction of the new requirement of the existing specification*, and not even the entire specification — for each new requirement the closest semantic group will be selected, and only inside her. At the beginning of the development of the requirements specification, the number of groups and the number of requirements within the groups will be very small, and by the time the specification has grown enough to have 10 requirements in each group, the old requirements will be tested long ago. This approach is carried out in approximately the same time frame as the previous one.

But you can go even further in thinking — if you introduce a *metric* according to which it will be possible to determine the distance between requirements, and those that are closer in semantics will be at a shorter distance than more distant ones. Then it is necessary to enter some *threshold value* and establish that if the distance between the requirements is below this value, then this means that the requirements contradict each other. In this case, it is possible to carry out a full check of all requirements. If you measure the distances within semantic groups, then you will need to personally check only those formulations of requirements, the distance between which is below a certain threshold value.

And here we come to the most difficult question: *how exactly to divide requirements into groups?* Different requirements can refer to different levels of decomposition, describe different functional features, and it is not always clear where one semantic group ends and another begins. If asked to divide requirements into groups, an examiner will most likely separate each N requirements of the specification into a separate structure, assuming that the specification is structured and sequential. However, conflicting requirements can be at different ends of the specification, and splitting them up in this way will not only get the project team closer to solving the problem, but rather wasteful validation process will take up valuable time. As for the idea of introducing a metric for requirements, this defies commentary, because the expert assessment in this case will be a subjective value, as well as the division into semantic groups is also very subjective.

Therefore, we need a tool devoid of any subjective view. This tool is an electronic computer, all value judgments of which have a mathematical basis.

And so, getting close to the value judgments made by the computer, we begin to dive into machine learning.

Machine learning (ML) addresses the question of how to build computers that improve automatically through experience [5]. In another way, it can be formulated like this — it is a process of searching for patterns in a large amount of information and making the best decision without human participation.

Machine learning is used in many areas of daily life. Smart "news feeds", contextual advertising, face recognition systems — all these are the results of ML algorithms. In addition, other experimental solutions are already being actively implemented, for example, the construction of a treatment regimen for a patient without human intervention. Definitely, in the future, the field of application of machine learning will only increase.

There are three main areas of ML [6]. The first thing we'll talk about is supervised learning. The tasks of this direction are reduced to approximately one thing: it is necessary to analyze the incoming labeled dataset and test some hypothesis. The second type of machine learning is deep machine learning — it can only work with what's called big data.

But we will be more interested in the third direction of machine learning — ML without a teacher. Unsupervised machine learning operates only with objects — class labels, one might say, are not available. And thus the algorithm must draw its own conclusions from the input datasets, without having any examples, in order to make decisions by analogy. Here are the tasks that ML solves without a teacher:

- clustering task;
- the task of finding association rules;
- anomaly detection task;
- dimensionality reduction problem;
- data visualization task.

The clustering task means dividing the original dataset into separate groups, called clusters, so that the objects of one cluster are similar, and the objects of neighboring clusters are different. The task of searching for association rules implies that the data is presented in the form of feature descriptions, and it is necessary from the entire array of features to find such feature values that are most often found in feature descriptions of objects. The task of finding anomalies is that the algorithm must determine an unusual set of data among all, that is, to detect that some objects behave in a fundamentally different way than expected from it. The problem of dimensionality reduction is posed when each object in the initial data has a large number of characteristics that determine it, and it is necessary to select from all of them only the most informative parameters that determine the behavior and, finally, the task of data visualization is used when it is necessary to somehow represent multidimensional quantities — in this case, objects from N-dimensional space are translated into one- two- or three-dimensional, after which they are reflected on a flat graph or in space.

Now let's go back to where we started — you need a tool that is devoid of a subjective view or is equally subjective in all cases, which will allow you to divide the specification of requirements into separate subgroups, and such that the requirements in different subgroups will differ in meaning, and in the same — be similar. We got the definition of clustering, in other words, according to our initial reasoning, it is necessary to cluster the requirements. And the unsupervised machine learning model just knows how to solve the clustering problem.

Thus, to summarize: to determine whether a new requirement contradicts the specification, it is necessary to cluster the specification and check the contradiction of the new requirement with those that are in the same cluster with it. In order to check the entire specification for consistency, it is necessary to cluster the specification, introduce the metric of the distance between the requirements within the cluster, and set the limit value at

which the requirements should be located without being conflicting, check all cases when the distance between the requirements is less than a given value. To do this, you need to understand how clustering works, which will be covered in the next section.

3. Methods for solving the clustering problem

When solving the clustering problem, we have N objects — $x_1 ... x_N$, each of which must belong to some class, as well as class labels $y_1 ... y_M$, $M \leqslant N$. For example, objects can be bicycles, or rather, their parametric description — weight, length, number of speeds, and class labels — the type of bicycle — children, sports, walking. Objects can be people of a specific height and weight, and class labels can be gender (male or female). Also, the objects can be requirements, and labels — the number of the semantic group, which the requirement belongs to. The task of clustering is to correctly place labels on objects in such a way that similar objects to each other fall into the same class, and objects that are fundamentally different — into different classes.

You can measure the quality of the clustering operation performed. We will definitely want the intra-cluster distance to be as small as possible and the distance between the clusters as large as possible. In mathematical terms, the formula is as follows: let F_0 be the average intra-cluster distance, and F_1 is the average distance between clusters. Then these values are calculated according to the formulas (3) and (4), and it is necessary to minimize their quotient, as shown in the formula (5):

$$F_0 = \frac{\sum_{i<j} [y_i = y_j] \rho(x_i, x_j)}{\sum_{i<j} [y_i = y_j]} \to \min. \quad (3)$$

$$F_1 = \frac{\sum_{i<j} [y_i \neq y_j] \rho(x_i, x_j)}{\sum_{i<j} [y_i \neq y_j]} \to \max. \quad (4)$$

$$\frac{F_0}{F_1} \to \min. \quad (5)$$

Clusters can be nested within each other. Depending on the level of decomposition, two objects can be located both in the same cluster or in different ones. With hard clustering, one object can be assigned to only one cluster; with soft clustering, it can be assigned to several classes with an indication of the weight. Clusters come in many different shapes, and fig. 1 shows an illustration of six different cluster views. Obviously, some methods do well with some forms, others with

Fig. 1. Various forms of clusters

others. And speaking of clustering methods, now we will analyze the main models.

The easiest way of clustering is k-means [7]. Objects that we clusterize are points in N-dimensional space. You must first select a number k — the number of clusters. It is assumed that it is known. Then all of a set of points you need to randomly select k pieces — this will be the centers of clusters. Then the distance of each point are measured for each center point is determined and in that cluster, the center of which it is closest. His new center, after which the operation is repeated as long as the change in position of the center of the cluster will not be substantially larger (smaller than a certain threshold value) for each of the resulting cluster is determined — then the cluster centers are recalculated. This method of k-means behalf Ball Hall. In the implementation of McKean cluster centers are recalculated whenever the object. When working with large data, not considering the arithmetic mean of all the objects of the cluster, it is taken from the random sample and the center is considered to be on it — it is an implementation of the Mini Batch. There is another implementation of k-means — the k-means++. In her initial approach is not chosen at random, and with a uniform distribution. And each successive approximation center of the cluster is set to be the most plausible in this situation. Fig. 2 (see the 2nd side of cover) shows an example of clustering added using this algorithm.

The result of the k-means method: in order to initiate the k-means object, you need to specify the number of clusters. The Mini Batch implementation allows you to work on a very large sample. The distance between the clusters is Euclidean. The k-means method is best used when the goal of the experiment is to identify clusters of approximately the same size.

The next clustering method is the EM (expectation—maximization) algorithm [8]. Let there be a sample of some random variable, and our task is to cluster the objects of this sample. The essence of the algorithm is in two steps — E and M. At step E, we need to define some hidden variables. At step M, depending on the values of the hidden variables, we calculate the probabilities of determining a particular object to a particular cluster and recalculate the hidden variables. Let our random variable be uniformly distributed, and then the hidden variables that we will determine at step E will be the mathematical expectation μ the value of the variance σ. Then, at step E, substituting certain μ and σ, we obtain the value of the probability density of attributing one or another object (x_i) to one or another cluster (j):

$$p_j(x_i) = \frac{1}{\sigma_i} \exp\left(-\frac{(x_i - \mu_j)^2}{2\sigma_j^2} \right). \quad (6)$$

Then, applying the Bayes formula, the probabilities of determining the i-th object to the j-th cluster are calculated:

$$g_{ji} = P(j \mid x_i) = \frac{w_j p_j(x_i)}{\sum_{k=1}^{K} w_k p_k(x_i)}. \quad (7)$$

At the second step (M-step), the likelihood maximization problem with fixed values of $P(j \mid x_i)$ is solved. If we equate the derivatives with respect to the parameters to zero, we obtain the expressions:

$$w_j = \frac{1}{N}\sum_{i=1}^{N} g_{ji},\ \theta_j = \arg\max_{\theta} \sum_{i=1}^{N} g_{ji} \ln \phi(\theta;\ x). \qquad (8)$$

At the same step, the hidden values are recalculated and the transition to step E is carried out. The cycle continues until convergence occurs (i. e., changes in the probabilities of assigning objects to different classes do not exceed some value specified by the user).

The clustering problem using the EM-algorithm is the problem of dividing a mixture of distributions: in which it is necessary to estimate the weights w_j and parameters θ_j of individual components. Figure 3 (see the 2nd side of cover) shows a visual reproduction of the application of the algorithm that was described above.

Summing up the results of EM clustering, to create an instance of a class, you need to specify the number of components. A model on a small sample can be retrained, i. e. get too close to a specific sample, and at the same time show a low result on test data, therefore, it is preferable to use it on a large sample. This model restores the distribution density; the resulting clusters are most likely to be convex. The distance between clusters is calculated in Euclidean metric.

The next technique for clustering is agglomerative hierarchical clustering [9]. First of all, let's define what is hierarchical clustering? This is clustering, in which clusters can be nested within each other. There are two fundamentally different methods for identifying such clusters:

- agglomerative — at the first stage, each object is placed into its own cluster, and at each next step, close clusters are combined into one.
- divisional — at the first stage, all objects are placed in one cluster, and at each next step, the cluster is split into smaller clusters.

The most common method is the first, and therefore in this work we will talk about agglomerative hierarchical clustering. This algorithm is preferable to use when we are talking about a strictly specified number of clusters — in this case, it is enough to wait until the objects are combined into a set of clusters, and the cardinality of the set will coincide with a fixed value.

In fact, this algorithm appears to be intuitive. The complexity can arise only when we think about how to calculate the distance between clusters? There are several possible ways. The distance between two clusters is the distance between the nearest neighbors of these clusters. Formalizing the above, we get:

$$R^{\text{Б}}(W,\ S) = \min_{w \in W,\ s \in S} \rho(w,\ s). \qquad (9)$$

Similarly, the distance between two clusters can be defined as the distance between the most distant neighbors of these clusters:

$$R^{\text{Д}}(W,\ S) = \max_{w \in W,\ s \in S} \rho(w,\ s). \qquad (10)$$

You can calculate the average distance between all objects of one and the other clusters:

$$R^{\text{С}}(W,\ S) = \frac{1}{|W||S|}\sum_{w \in W}\sum_{s \in S} \rho(w,\ s). \qquad (11)$$

Another way is to calculate the centers of the clusters and consider the distance between them as the distance between the clusters:

$$R^{\text{Ц}}(W,\ S) = \rho^2\left(\sum_{w \in W} \frac{w}{|W|},\ \sum_{s \in S} \frac{s}{|S|}\right). \qquad (12)$$

Finishing with the problem of measuring the distance between clusters, we propose another possible measure — the Ward distance:

$$R^{\text{У}}(W,\ S) = \frac{|S||W|}{|S| + |W|}\rho^2\left(\sum_{w \in W} \frac{w}{|W|},\ \sum_{s \in S} \frac{s}{|S|}\right). \qquad (13)$$

Figure 4 shows a step-by-step view of the agglomerative hierarchical clustering process.

Summing up the discussion of agglomerative hierarchical clustering, to initialize the model, it is necessary to set the number of clusters, the method for calculating the distance between clusters from those described above, and the metric. This method shows itself well in the case of a large number of objects, and in the case of a large number of clusters — in fact, in this situation the model demonstrates the best results, and any metric can be set.

We now turn to the final clustering model, the density-based model DBSCAN [10]. Such models work on the assumption that objects are a set of points distributed with certain densities, and in the vicinity of some points there are other points. Let there be some point (object) consider its neighborhood of radius R. If there are N or more other point objects in the specified neighborhood, then such a point is the main one. If in the vicinity of points there are less than N, but there is at least one main point, then it will be called a boundary point. In all other cases, it is noise.

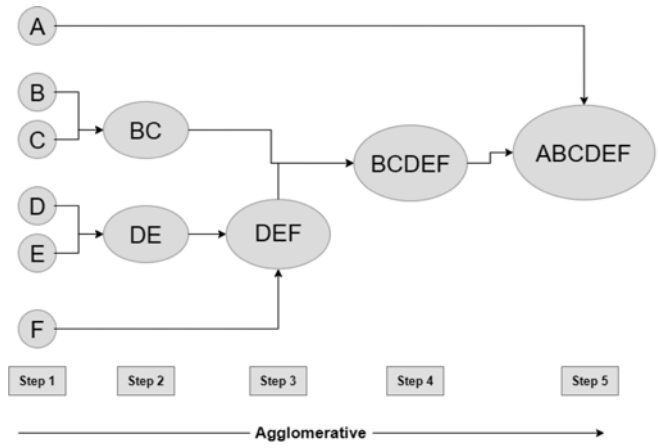

Fig. 4. Agglomerative clustering

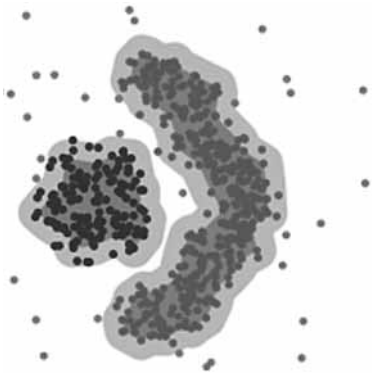

Fig. 5. DBSCAN Clustering

The DBSCAN algorithm is as follows.

- It is necessary to assign each point to one of the classes (main, borderline, noise).
- Connect the points at a distance L to each other — the result of the work is a graph.
- Combine all main points connected by a line into one cluster — in other words, select the components of the graph's connectivity.
- Assign the remaining border points to the corresponding clusters.

In practice, DBSCAN shows high results on complex shapes of clusters, it perfectly detects noises that pretty much spoil other clustering algorithms. However, when dealing with simple cluster shapes that are close to each other, DBSCAN can be wrong. Figure 5 shows the result of this algorithm.

Summing up the conversation about DBCAN, to initialize the model, it is necessary to set the radius of the neighborhood in which to search for nearby points, and the number of neighbors that the point must have in order for it to become the main one. The algorithm shows good results on a large number of objects and a small number of clusters and does an excellent job in identifying non-trivial forms. The distance between clusters is calculated in Euclidean metric.

4. Methods for vector representation of sentences

Let's start with the simplest method, the simple cosine distance method [11]. The first step of the algorithm is to tokenize sentences into words. The next step is lemmatization, i. e. reducing all words to their normal forms and excluding everything except words. The third step of the algorithm is to determine the set of words for all these sentences. Let us define a vector space **V** of dimension n, where n is the cardinality of the set, and assign to each word from the set some vector in **V** so that the obtained vectors constitute a system of pairwise orthogonal vectors. Thus, we get that the j-th coordinate of the i-th word will be equal to:

$$a_{ij} = \begin{Bmatrix} 1 | i = j \\ 0 | i \neq j \end{Bmatrix}. \qquad (14)$$

The fourth step of the algorithm is to determine for each sentence of our text its vector in the space **V**. This procedure can be described in simple words: we sequentially consider each word of the "bag of words" [12], if the i-th word of the dictionary is present in the sentence, then its i-th the coordinate of the vector of the corresponding sentence takes the value of one, and otherwise — zero. This procedure can be clearly seen at table 1.

Table 1

Definition of vectors from word space

Sentence ID	Value	Division	Timer	Shall	Be	Second	Error	Measure	Time
1	1	1	1	1	1	1	0	0	0
2	0	0	1	1	1	1	1	0	0
3	0	0	1	1	0	0	0	1	1

The final stage of the algorithm is to select the sentence for which we want to determine the closest sentence in meaning, determine the cosine distances between the vector corresponding to the target sentence and all other vectors. The minimum cosine distance means the greatest semantic similarity. According to the formula (15) the order of calculating the cosine distances between vectors is determined:

$$\rho_{\cos}(x,\ y) = \arccos\left(\frac{\langle x,\ y\rangle}{\|x\| \cdot \|y\|}\right) = \\ = \arccos\left(\frac{\sum\limits_{i=1}^{d} x_i y_i}{\sqrt{\sum\limits_{i=1}^{d} x_i^2} \cdot \sqrt{\sum\limits_{i=1}^{d} y_i^2}}\right). \qquad (15)$$

Applying the resulting formula to the vectors above, we get that $\rho_{\cos}(a_1,\ a_2) \approx 0.73$, $\rho_{\cos}(a_1,\ a_3) = 0.4$, $\rho_{\cos}(a_2,\ a_3) \approx 0.45$. The algorithm determined, as expected, the first and second sentences as the closest pair, and the first and third sentences as the farthest. But this trivial approach only works with simple examples like the one above. With regard to the practice of developing requirements, in the specification it is quite likely that requirements of the form: "The walls of the building should be painted blue", "The ceiling of the building should be painted white". Obviously, the algorithm described above will take into account the almost identical filling of both sentences with words, and will issue a recommendation to check these requirements for possible inconsistency. However, there is no contradiction, because the key and most important detail in this proposal is not the description of the function (should be painted), but the object that this function is aimed at – the walls or ceiling. Therefore, the transformation of sentences into numerical vectors based only on the presence or absence of certain words is obviously fraught with problems. Moreover, this algorithm in no way took into account the frequency of occurrence of words, but took into account the words "common" for all requirements, which should not be.

Regarding the last remark, of course, there is a high-quality add-on for this algorithm. It's about TF-IDF Vectorizer.

TF-IDF (Term Frequency Inverse Document Frequency) [13] is a very common sentence-to-numeric vector algorithm used in unsupervised machine learning. TF-IDF is an algorithm that determines a measure of the originality of a word by comparing the number of times a word appears in a document with the number of documents in which that word appears. The calculation of the originality of a word is performed according to the following formula:

$$TF\text{-}IDF = TF(t,\ d) \times IDF(t) = \\ = TF(t,\ d) \times \log \frac{1+n}{1+df(d,\ t)} + 1, \quad (16)$$

where $TF(t,\ d)$ is the frequency of the word, the number of times the word t occurs in the document d, is the reciprocal frequency of the document, calculated according to n — the number of documents, and $df(d,\ t)$ — the number of documents in which the given word occurs.

The algorithm is as follows — at the first step, you also need to create a dictionary based on sentences (documents), having previously reduced them to normal form. We will consecutively number all the words that occur in sentences. Stop words (for example, for the Russian language these are conjunctions, prepositions) will not be counted and numbered, since they do not represent any information. The second step is to transform the sentence space into a vector space. We define the following function $TF(t,\ d)$:

$$TF(t,\ d) = \sum_{x \in d} FR(x, t);\ FR(x,\ t) = \begin{cases} 1, x = t \\ 0, x \neq t \end{cases}. \quad (17)$$

As a result, sentences $d_1 \ldots d_m$ in the word space $t_1 \ldots t_n$ will have the following form:

$$\vec{v}_{d_i} = \left(TF(t_1,\ d_i),\ TF(t_2,\ d_i),\ \ldots,\ TF(t_n,\ d_i)\right). \quad (18)$$

Combining the vectors of the sentence space into a matrix, we get a matrix $TF_{m \times n}$. The next step is to calculate IDF using formula (17), as a result of which the resulting vector of dimension n is reduced to the form of a diagonal matrix IDF of size $n \times n$. At the final stage, the matrices TF and IDF are multiplied, and the obtained normalized result is a matrix, along the rows of which the sentence vectors are located in the numerical space. This result is somewhat more meaningful than the method of simple cosine distances, it takes into account the frequency of words both within a sentence, as in other sentences of the text, but the algorithm still depends on the presence of a particular word in the sample and does not take into account different formulations of the same essence.

The latest Google development is the Word2Vec algorithm [14], which transforms words into the space of numeric vectors, taking into account the semantics of words. There are two fundamentally different approaches to Word2Vec — CBOW and skip-gram. Let's take a look at each of these approaches in turn.

The CBOW (Continuous bag of words) method allows you to predict words based on context. Suppose you have an input sentence "It is raining outside the window." The CBOW method will read the words "for", "window", "rain" and will try to predict the word "pouring". Words in Word2Vec are represented as N-dimensional vectors. A pretrained neural network is used to transform a word as a set of letters into an N-dimensional vector. With each prediction of a word from the context, the neural network is retrained and determines the value of the new N-dimensional vector as a new word.

The skip-gram method, in another case, allows one to predict the context one word at a time. For example, using the word "pouring" to restore the words "for", "window", "rain". This method is somewhat slower, but works better with words that are infrequent. It also uses a neural network.

The result of the work of any of the Word2Vec algorithms are N-dimensional vectors that define the words of the sentence. The advantage of Word2Vec over the TF-IDF algorithm described above is that TF-IDF vectors have a dimension that matches the cardinality of a dictionary per document — and this is an order of more than 100,000, while the dimension of Word2Vec vectors ≈ 300. Therefore, the resulting vectors can be used in supervised machine learning methods, for example, in the classification problem, if we consider each coordinate of the vectors as a feature value. But the main advantage of Word2Vec is the restored semantic relations between the words of the dictionary. The resulting numeric representations of words allow relational operations on words. The most striking and recognizable example of Word2Vec sounds like this: the difference between the words "king" and "queen" is equal to the difference between the words "man" and "woman", i. e. "King — queen = man — woman".

However, when working with requirements, it is not enough for us to be able to transform individual words into vectors — it is necessary to do the same with whole sentences. The first intuitive and method that comes to mind will be averaging all the word vectors in a sentence, but this approach does not take into account the change in word order in any way, so let's delve deeper into the search for a technology that extends Word2Vec to the entire document — and such a technology exists and is called Doc2Vec.

According to [15], the Doc2Vec approach can be described as follows — each sentence is transformed into a numeric vector, which is a column in the matrix D, each word is also transformed into a numeric vector according to the well-known Word2Vec algorithm, which is a column in the matrix $\mathbf{W}$. Thus, in addition to use only vectors of words to predict another word, we additionally introduce a sentence vector that is unique for each sentence. Thus, the result of training the neural network is the sentence vector we are looking for.

As in the case of Word2Vec, there are two approaches to defining the sentence vector — DM (distributed memory) and DBOW (distributed bag of words). DM allows you to define a word from a context, DBOW — a context from a sentence vector.

Summing up the consideration of this model, we can say with confidence that doc2vec is a new step towards

the development of technology for transforming sentences into numerical vectors. However, it also has a number of disadvantages, first of all, it is the dependence on the independent learning of the neural network, and this requires rather large data. From the articles on the implementation of doc2vec, it was found that if you train doc2vec on data of 100,000 sentences, then the accuracy of the results of the regression algorithms, classification, using the vector coordinates obtained as a result of the doc2vec algorithm as a feature, is only 74...76 %. Certainly, it cannot be argued that if the supervised machine learning models show such a mediocre result, then the clustering of the obtained vectors will be insignificantly better, but this makes one think. And returning to the context of the application of algorithms — the specification of requirements — you involuntarily ask yourself the question "will there be 100,000 proposals in the specification of the requirements of any industrial project?" Of course not, not in all. But definitely, the assumption that the doc2vec model will perform poorly on small amounts of data should be tested.

The final model to be discussed in this paper is BERT [16]. The BERT model was announced and made public in 2018. In addition to the fact that the algorithm demonstrated the highest results of work, the BERT developers provided already trained data models on large datasets, and thus relieved potential system developers who decided to use BERT from training problems. It is enough to import the finished model into the code, after which it is possible to start development.

Initially, the BERT model is trained on a huge amount of text taken from books, Wikipedia, etc., and in several languages. This technology is based on such algorithms developed by the NLP community as learning with partial involvement of a teacher, ELMo models (Embeddings from Language Models), ULMFiT (Universal Language Model Fine-Tuning), OpenAI Transformer and others.

BERT is based on the Transformer, in fact, BERT is a trained stack of Transformer encoders. Encoders are objects of the same structure, but with different weights. Each encoder consists of two parts — an inner understanding layer and a propagation layer. The encoder receives as input numerical vectors obtained from words, processes the vectors at the level of the internal understanding layer and transmits them to the feedforward neural network. The result of the output of one encoder level is the input for the next encoder. Encoders form a stack of 12 (24 for deeply trained models) elements, an encoder stack, and is the backbone of BERT, as mentioned. Each sent word in BERT goes through a chain of encoders and as a result, for each word, a numeric vector of dimension 768 is formed for the base BERT implementation. The obtained vectors are the required ones in the framework of this problem. Further, it is possible to perform various operations with them: cluster, use as input data in supervised machine learning algorithms, for example, for a logistic regression problem.

Let's note some more unique features of BERT. BERT is not a single directional model, it means that it does not perceive words sequentially from left to right, for example. Such a method definitely allows one to take into account the context of words without reference to two neighbors. Obviously, the most difficult stage in the life of a BERT is the prediction stage, which corresponds to the results of the development of the encoder propagation layer. Whereas conventional models predict words from left to right, BERT avoids this solution in the following way: before reading a list of words, a BERT component called a marker replaces 15 % of each line, thus masking the terms. After that, BERT tries to guess the changed words in accordance with the context in which the marker did not work. To predict a specific word, you first need to introduce a word classification tool at the encoder output and calculate the probability of the presence of each word using softmax.

By receiving different words from the same sentence as input, BERT tries to predict whether the second word is a logical continuation of the first, assuming that half of all input is pairs and the remaining half of the words are random. In order for the model to understand the difference between the two words during training, the CLS indicator is inserted at the beginning of each word, and the SEP indicator at the end.

To summarize the description of BERT, it is an extremely efficient model for NLP problems and extremely difficult to understand. The presence of an already trained neural network of several hundred hidden levels allows you to obtain the highest results in solving the problem, without spending computer resources and time resources. And multilingualism allows you to work with the semantics of sentences in a variety of languages.

5. Results

This chapter presents the results of applying the algorithms described above to test data. As test data, a specification of requirements from 282 requirements in Russian was used, and several of them were artificially added to create inconsistencies that will need to be found by our algorithms. Therefore, two groups of contradictions were created, each with three requirements. The first group contradicts in terms of the timer division value, and the second — in terms of errors:

R-0224: Цена деления таймера должна составлять 1 минуту

R-0278: Цена деления шкалы таймера 1 мин

R-0282: Цена деления шкалы таймера 2 мин

R-0238: Погрешность таймера должна составлять 0,5 мин.

R-0221: Погрешность таймера должна составлять 1 мин.

R-0277: Таймер должен производить отсчет времени с погрешностью не более 0.5 мин

As can be seen from these contradictions, a couple of sentences in theory should be defined very simply — they simply change the number corresponding to either the division price or the error. For each group, a contradiction was introduced with a reformulation — if the system is able to identify such contradictions too, this can be regarded as a success.

First, let's try to compare clustering models with each other. Let us fix the TF-IDF algorithm as an algorithm for obtaining vectors from sentences and successively try to determine the cluster for the requirements R-0282,

	ID	Requirement		ID	Requirement
221	R-0224	Цена деления таймера должна составлять 1 минуту	209	R-0212	Рабочий угол таймера должен составлять 360 гра...
275	R-0278	Цена деления шкалы таймера 1 мин	218	R-0221	Погрешность таймера должна составлять 1 мин.
279	R-0282	Цена деления шкалы таймера 2 мин	235	R-0238	Погрешность таймера должна составлять 0.5 мин.

Fig. 7. The result of the work of k-means

R-0238. Let's start by checking the k-means clustering, which is visualized in the fig. 6 (see the 3rd side of cover).

We will evaluate the result of clustering (both this and the others) as follows: all conflicting requirements must be in one cluster, other requirements either should not exist at all, or their number must be minimal. In the case of k-means, we get clusters (fig. 7).

In such a degenerate case, the algorithm was able to identify in one cluster all the contradictions of group 1 (3/3) and nothing more. No other requirements that do not tell about the division price of the timer scale were not found in this cluster. However, for the second group, the circumstances were not so successful. We managed to identify two (2/3) contradictions, but instead of the third, a "normal" requirement was added to the group. Thus, the algorithm made a mistake once.

Now consider the EM algorithm, the clustering of which is presented at fig. 8 (see the 3rd side of cover).

The EM algorithm specializes in separating mixtures. Indeed, if we take a close look at the two images, comparing them with k-means, we can see how EM is trying to divide some volumetric clusters into several (perhaps even when it is not needed). For example, the cluster of points at the coordinate (−20, 20) was determined by k-means in one cluster, while EM tried to restore the distribution over mixed clusters. When trying to identify inconsistencies, the EM-algorithm correctly identifies group 1 (3/3, 0 false), however, compared to the previous one, it adds two extra requirements to the second cluster, determining only two true (2/3, 3 false) (fig. 9).

Probably, the test specification of the requirements does not contain a mixture of clusters, and therefore the legitimacy of the use of clustering, which specifies the separation of mixtures, is called into question.

The next clustering model is agglomerative hierarchical clustering, which aims to separate nested clusters. The first option is that the distance between clusters is the average distance between all elements.

The fig. 10 (see the 3rd side of cover) shows just such a variant of hierarchical clustering. As for the results, then (3/3, 0 false) for the first group, (2/3, 4 false) for the second (fig. 11). Again, the first group can be dealt with quite easily, while the errors for the second only grow. Apparently, there are no hierarchical clusters in the specification.

If we begin to define the distance between clusters as the smallest distance between its elements, then such clustering will lead to the situation at fig. 12, see the 3rd side of cover. Already from the rendered picture, it becomes clear that no positive results can be expected from such a setting — the mechanism for determining the boundaries of clusters, based on the position that the distance between the clusters coincides with the distance of the nearest elements, leads to the fact that the clusters collapse very quickly, and that's it. the timer requirements fell into one large cluster of 80 objects. There is no sense in such clustering.

	ID	Requirement		ID	Requirement
221	R-0224	Цена деления таймера должна составлять 1 минуту	209	R-0212	Рабочий угол таймера должен составлять 360 гра...
275	R-0278	Цена деления шкалы таймера 1 мин	218	R-0221	Погрешность таймера должна составлять 1 мин.
279	R-0282	Цена деления шкалы таймера 2 мин	229	R-0232	Коэффициент оптического контраста циферблата п...
			235	R-0238	Погрешность таймера должна составлять 0.5 мин.
			276	R-0279	Контрастность шкалы к корпусу таймера должна с...

Fig. 9. The result of the EM algorithm

	ID	Requirement		ID	Requirement
221	R-0224	Цена деления таймера должна составлять 1 минуту	209	R-0212	Рабочий угол таймера должен составлять 360 гра...
275	R-0278	Цена деления шкалы таймера 1 мин	218	R-0221	Погрешность таймера должна составлять 1 мин.
279	R-0282	Цена деления шкалы таймера 2 мин	229	R-0232	Коэффициент оптического контраста циферблата п...
			235	R-0238	Погрешность таймера должна составлять 0.5 мин.
			236	R-0239	Корпус таймера должен иметь гидрофобное покрытие
			276	R-0279	Контрастность шкалы к корпусу таймера должна с...

Fig.11. The result of the Agglomerative Average

If we go from the opposite, to maximize the distance between the clusters, then the algorithm will try to "pull apart" the clusters as far from each other as possible (fig. 13, see the 3rd side of cover).

And already these results may be of interest to us. The algorithm made a mistake once in the search for group (3/3, 1 false), and three times in the second group (2/3, 4 false). As you can see (fig. 14), this setting of agglomerative clustering is an order of magnitude better, however, it is far from perfect, which raises a natural question — is there any point in implementing it.

The latest implementation of this algorithm is Agglomerative Ward. The model tries to minimize the number of combined clusters, and in theory, the picture observed with this algorithm should be the opposite of Agglomerative Single (fig. 15, see the 4th side of cover).

The result of this algorithm is shown in the fig. 16. The totals table looks pretty good: (3/3, 1 false), (2/3, 1 false). In general, this result is comparable to the k-means algorithm.

If we compare all the implementations of agglomerative clustering presented above, we can say that plus or minus the results are similar (if we do not take into account, of course, the algorithm in which the distance was measured as the minimum distance between cluster objects). However, one cannot achieve one hundred percent result, perhaps nested clusters are also not a picture that corresponds to the test specification.

And finally, the DBSCAN algorithm was considered (fig. 17, see the 4th side of cover).

Frankly speaking, it was clear in advance that this algorithm would not give the desired result — since it is based on the density of a distributed quantity, and in our case it cannot be argued that the density will be constant anywhere. Therefore, it turned out that again a large number of requirements were in one cluster, and such clustering is not informative. Yes, the user will learn the obvious — all requirements for a timer belong to one semantic group. However, finding contradictions among the 83 requirements is a very non-trivial task, and it cannot be said that in this case the goal of the system will be achieved, because the user will not significantly reduce the time for checking the requirements specification.

An intermediate result can be summed up: after testing all clustering options on the TF-IDF model, it turned out that the k-means method would be the best option. However, this method did not quite correctly identify group 2, so we will try to replace the model for transforming sentences into vectors, although it is worth noting here that TF-IDF has shown itself very well at the level of trivial examples.

The first option is the simple cosine distance method (fig. 18, see the 4th side of cover).

This method is a lightweight version of the TF-IDF algorithm. It does not take into account repetitions, it does not take into account "garbage" words, i. e. those words that occur in most sentences, and of course, it cannot be taken seriously. The picture in the figure above does not imply clustering at all — the points are almost evenly distributed on the surface, and no clusters are visually observed. Even on the basis of visual observation, it can be concluded that this model is unsuitable — and the results confirm this assumption. 43 requirements end up in the cluster, where group 1 was lucky enough to get, and 22 requirements, respectively, in group 2. This is one of those models that cannot be used in any form and was presented here purely out of its simplicity to facilitate understanding of the process.

The next model is doc2vec (fig. 19, see the 4th side of cover). And here we are faced with the first surprise — doc2vec does not show results. Regardless of whether to lemmatize sentences or not, whether to expose a distributed memory method or bags of words, the results leave much to be desired. Here is one option — DBOW (fig. 20).

Yes, both clusters speak about one thing — about the timer, but in the first case we have not a single revealed contradiction (1/3, 14 false) and a huge number of requirements that do not contradict the given one. In the second case, the picture is somewhat more pleasant — two

	ID	Requirement		ID	Requirement
220	R-0223	По диаметру таймера должны быть расположны де...	155	R-0158	Соотношение ширины полос в соседних группах до...
221	R-0224	Цена деления таймера должна составлять 1 минуту	209	R-0212	Рабочий угол таймера должен составлять 360 гра...
275	R-0278	Цена деления шкалы таймера 1 мин	218	R-0221	Погрешность таймера должна составлять 1 мин.
279	R-0282	Цена деления шкалы таймера 2 мин	229	R-0232	Коэффициент оптического контраста циферблата п...
			235	R-0238	Погрешность таймера должна составлять 0.5 мин.

Fig. 14. Result of Agglomerative Complete

	ID	Requirement		ID	Requirement
220	R-0223	По диаметру таймера должны быть расположны де...	209	R-0212	Рабочий угол таймера должен составлять 360 гра...
221	R-0224	Цена деления таймера должна составлять 1 минуту	218	R-0221	Погрешность таймера должна составлять 1 мин.
275	R-0278	Цена деления шкалы таймера 1 мин	235	R-0238	Погрешность таймера должна составлять 0.5 мин.
279	R-0282	Цена деления шкалы таймера 2 мин			

Fig. 16. The result of the Agglomerative Ward

	ID	Requirement		ID	Requirement
21	R-0022	СрЗИ не должны ухудшать функциональные характе...	210	R-0213	У таймера должна быть возможность дозаводки
217	R-0220	Срок эксплуатации таймера должен составлять не...	213	R-0216	Нижняя часть таймера должна быть аппаратной
220	R-0223	По диаметру таймера должны быть расположны де...	214	R-0217	Верхняя часть таймера должна быть подвижной
223	R-0226	На циферблате должны быть использованы арабски...	218	R-0221	Погрешность таймера должна составлять 1 мин.
224	R-0227	Циферблат должен располагаться на нижней части...	231	R-0234	Таймер должен быть цилиндрической формы
230	R-0233	Цвет стрелки должен соответствовать цвету цифе...	235	R-0238	Погрешность таймера должна составлять 0.5 мин.
232	R-0235	Диаметр таймера не должен превышать 6 см	238	R-0241	У таймера должна быть нижняя часть
233	R-0236	Высота таймера не должна превышать 7 см	269	R-0272	Таймер должен заводиться поворотом ручки
236	R-0239	Корпус таймера должен иметь гидрофобное покрытие	270	R-0273	Таймер должен отсчитывать время
247	R-0250	У таймера должна быть защита механизма от внеш...			
263	R-0266	Звук отсчета времени должен быть слышен (посек...			
266	R-0269	Габариты таймера не должны превышать 10x10x10 ...			
273	R-0276	Таймер должен иметь возможность досрочного отк...			
279	R-0282	Цена деления шкалы таймера 2 мин			

Fig. 20. Doc2Vec DBOW Results

	ID	Requirement
206	R-0209	Длительность звукового сигнала должна составля...
218	R-0221	Погрешность таймера должна составлять 1 мин.
221	R-0224	Цена деления таймера должна составлять 1 минуту
235	R-0238	Погрешность таймера должна составлять 0.5 мин.
274	R-0277	Таймер должен производить отсчет времени с пог...
275	R-0278	Цена деления шкалы таймера 1 мин
279	R-0282	Цена деления шкалы таймера 2 мин

Fig. 22. Result of BERT

contradictions have been found, but the errors are still significant (2/3, 7 false).

What is the reason for this? The situation is, in fact, extremely simple — we are using the untrained doc2vec model. First, it needs to be trained on our requirements, and 280 pieces are negligible, even if you use cross-validation. This is why the model lacks the information to build a vocabulary, and therefore the results come out so unimpressive. But here it is impossible to operate only with the fact that the specification is not voluminous enough. There are also small projects, and in such cases the number of requirements is estimated at several hundred, which means the algorithm will not work. We need to take an already trained model for our calculations.

And there is such a model — we will use BERT trained on Wikipedia data, etc. It is enough to load the already trained neural network (moreover, it is configured just to search for duplicates), and you can start getting vectors for sentences from it (fig. 21, see the 4th side of cover).

Already from the illustration there is a feeling that the results will be successful. In the picture above, clusters are visible, the points do not overlap each other strongly enough, which is a good signal. The search for clusters brings amazing results: both groups end up in the same cluster and in full. This algorithm was able to determine both simple contradictions associated with a change in one digit in the text of the requirement, and reformulation. This is the first algorithm that determined (6/6) with only one error — BERT added the R-0209 requirement to the cluster, which does not contradict anything (fig. 22).

Conclusion

In the process of writing the article, an analysis was made of clustering methods and methods for converting sentences into vectors.

The analysis was carried out as follows: contradictions were deliberately introduced into the requirements specification, after which all possible algorithms for converting sentences into vectors with clustering models were launched. As a result, the most successful options were identified that are suitable for use in industrial requirements development. The table 2 illustrates the possibility of using a combination of the two models. The

NLP-clustering intersections that are not recommended for work are marked in black, the combinations that have demonstrated high results are in striped.

Table 2

Applicability of models for verification of requirements

Vectorizing Algorithm	Clustering Algorithm						
	k-means	em	aggloaver	agglomin	agglomax	aggloward	dbscan
cosine							
TF-TDF							
doc2vecdbow							
doc2vecdm							
BERT							

In the future, it is planned to continue working on the topic. The implementation of DeepPavlov [17], Fasttext [18] and the search for other models for converting sentences into numeric vectors are already planned. Already from this work, it became clear that models based not on the frequency of the presence of words in a sentence, but on deeply trained neural networks, will show themselves best. Therefore, an additional analysis of the available options is required. It is also possible to find a pre-trained doc2vec model, or to conduct an analysis on a specification consisting of several thousand requirements and achieve a positive result.

With regard to the interface, it is planned to develop a plugin for Google Docs. It often happens that the requirements are stored in the cloud as a table in a .docx file. Google Docs' Check Requirement button would make life easier for analysts, if only because it would not require them to upload requirements specification into.csv format.

References

1. **Hall E.** *Requirements Engineering*, US, Kent, Gray Publishing, 2005, 239 p.

2. **Batovrin V., Gaydamaka K.** Requirements engineering — key factor for project success, *The Project Management Journal*, 2017, no. 1 (49), pp. 6—20.

3. **Chatzipetrou P., Unterkalmsteiner M., Gorschek T.** Requirements' Characteristics: How do they Impact on Project Budget in a Systems Engineering Context? *In2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* 2019 Aug 28, IEEE, 2019, pp. 260—267.

4. **INCOSE,** Guide for writing requirements INCOSE TP-2010-006-02, USA, San Diego, International Council on Systems Engineering, 2015, 73 p.

5. **Jordan M. I., Mitchell T. M.** Machine learning: Trends, perspectives, and prospects, *Science*, 2015, vol. 349, no. 6245, pp. 255—260.

6. **Muhamedyev R.** Machine learning methods: An overview, *Computer modelling & new technologies*, 2015, vol. 19, no. 6, pp. 14—29.

7. **Hartigan J. A., Wong M. A.** Algorithm AS 136: A k-means clustering algorithm, *Journal of the royal statistical society. Series C (applied statistics),* 1979, vol. 28, no. 1, pp. 100—108.

8. **Moon T. K.** The expectation-maximization algorithm, *IEEE Signal processing magazine,* 1996, vol. 13, no. 6, pp. 47—60.

9. **Zhao Y., Karypis G., Fayyad U.** Hierarchical clustering algorithms for document datasets, *Data mining and knowledge discovery*, 2005, vol. 10, no. 2, pp. 141—68.

10. **Schubert E., Sander J., Ester M., Kriegel H. P., Xu X.** DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems,* 2017, vol. 42, no. 3, article 19, pp. 1—21.

11. **Sravanthi P., Srinivasu B.** Semantic similarity between sentences, *International Research Journal of Engineering and Technology (IRJET),* 2017, vol. 4, no. 1, pp. 156—61.

12. **Zhang Y., Jin R., Zhou Z. H.** Understanding bag-of-words model: a statistical framework, *International Journal of Machine Learning and Cybernetics*, 2010, vol. 1, pp. 43—52.

13. **Ramos J.** Using TF-IDF to determine word relevance in document queries, *Proceedings of the first instructional conference on machine learning 2003*, 2003, 4 p.

14. **Rong X.** word2vec parameter learning explained. arXiv preprint arXiv:1411.2738. 2014 Nov 11.

15. **Lau J. H., Baldwin T.** An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv preprint arXiv:1607.05368. 2016 Jul 19.

16. **Devlin J., Chang M. W., Lee K., Toutanova K.** Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. 2018 Oct 11.

17. **Burtsev M., Seliverstov A., Airapetyan R., Arkhipov M., Baymurzina D., Bushkov N., Gureenkova O., Khakhulin T., Kuratov Y., Kuznetsov D., Litinsky A.** DeepPavlov: Open-source library for dialogue systems, *Proceedings of ACL 2018*, System Demonstrations 2018 Jul., 2018, pp. 122—127.

18. **Joulin A., Grave E., Bojanowski P., Douze M., Jégou H., Mikolov T.** Fasttext. zip: Compressing text classification models. 2016. arXiv preprint arXiv:1612.03651.

УДК 004.052.42 DOI: 10.17587/prin.13.200-208

И. А. Федотов[1], аспирант, ivan.fedotov@phystech.edu,
А. С. Хританков[2], канд. физ.-мат. наук, доц., akhritankov@hse.ru,
М. Д. Обидаре[1], студент, obidarefolu1@gmail.com
[1] Московский физико-технический институт,
[2] Национальный исследовательский университет "Высшая школа экономики", Москва

Автоматическая верификация многосторонних соглашений и планирование отправки сообщений в системах распределенного реестра

Многосторонние соглашения используются в системах распределенного реестра и блокчейн-сетях для согласования изменений в системе. Если один из участников сети предлагает транзакцию на запись, то сначала ее должны подтвердить определенные участники сети. Многостороннее соглашение, или консенсус, определяет состав этих участников. На основе предыдущих ответов можно посчитать вероятность подтверждения транзакции для каждого из участников. В настоящей работе предложен метод статистической проверки моделей для определения вероятности того, что консенсус будет достигнут. Отправка запросов на подтверждение может требовать дополнительных затрат. Кроме отмеченной вероятности вычислено математическое ожидание числа сообщений, которые прошли в сети до достижения консенсуса. Приведена модель или несколько моделей консенсуса в виде марковской цепи с различными стратегиями отправки сообщений. На основе алгоритмов построения модели и спецификации разработано инструментальное средство анализа консенсуса и отправки сообщений на подтверждение.

Ключевые слова: *консенсус, блокчейн, верификация, статистическая проверка моделей, марковская цепь*

Введение

Технология распределенного реестра получила в последнее время широкое распространение. Хранилища на основе технологии блокчейн используются в различных областях: юриспруденция; отслеживание активов; финансовый сектор и ряд других [1]. Блокчейн удобен при использовании в системах, где несколько заинтересованных сторон, не доверяющих друг другу, должны достигнуть соглашения. Консенсус утверждает состав участников, которые должны подтвердить транзакцию, чтобы она считалась валидной для всех участников сети [2].

Каждый участник сети может либо подтвердить, либо отклонить транзакцию. Так как ответ каждого участника заранее неизвестен, то процесс подтверждения транзакции является вероятностным. На основе полученных ответов от каждого участника консенсуса можно вычислить вероятность того, что в следующем раунде подтверждения участник консенсуса подтвердит транзакцию. Условия консенсуса могут задаваться в виде логической формулы. Таким образом условия консенсуса задаются во фреймворке Hyperledger Fabric (HLF), который является одной из самых используемых платформ для построения блокчейн-систем [3]. Предложенные в статье алгоритмы и инструментальные средства могут использоваться для верификации правил одобрения HLF.

Уязвимости в блокчейн-системах приводили к значительным потерям [4]. В частности, уязвимости в принятых правилах одобрений HLF являлись причиной финансовых потерь [5]. Верификация многосторонних соглашений позволяет выявить уязвимости до развертывания соглашений в сети.

В исследовании, результаты которого представлены в настоящей статье, представлено построение модели консенсуса в виде марковской цепи. Условия консенсуса передаются в виде вероятностной линейной временной логики (*probabilistic linear temporal logic*, pLTL) [6]. Использован метод статистической проверки моделей [7] для определения вероятности достижения консенсуса. Использованы различные уникальные комбинации последовательностей отправки сообщений на подтверждение. В целом, результаты предлагаемого авторами статьи подхода состоят в следующем:

- предложен алгоритм построения марковской цепи и спецификации для консенсуса, заданного в виде pLTL-формулы;

- разработан модуль автоматического построения модели и проверки спецификации;
- предложен алгоритм отправки сообщений на подтверждение на основе выбранной модели марковской цепи;
- разработан модуль автоматической отправки сообщений на подтверждение;
- представлена возможность интеграции разработанного инструментального средства с фреймворком Hyperledger Fabric.

Смежные работы

Формальные методы верификация и тестирования применяются в процессе разработки и отладки блокчейн-систем [8]. Верификация с помощью метода статистической проверки моделей хорошо изучена, и для этого могут использоваться различные инструментальные средства [9]. В недавних исследованиях [10, 11] авторы продемонстрировали возможность использования статистической проверки моделей для верификации блокчейн-систем. Модель в виде марковской цепи и спецификация в виде pLTL-формулы может быть представлена для протоколов взвешенного консенсуса [12]. В работе [12] при этом не исследован вопрос модификации модели консенсуса, если вероятность принятия транзакции недостаточна для пользователя. Более того, результат верификации может быть применен для автоматического режима планирования и отправки сообщений для достижения консенсуса. Интересен вопрос интеграции инструментальных средств верификации консенсуса с существующими платформами построения блокчейн-сетей. В настоящей статье рассмотрены перечисленные открытые вопросы.

Технология распределенного реестра и метод статистической проверки моделей

В настоящем разделе представлены протоколы консенсуса, которые впоследствии будут верифицированы. Также рассмотрены методы и инструментальные средства статистической проверки моделей, которые используются для верификации многосторонних соглашений.

Многосторонние соглашения в блокчейн-системах. Блокчейн является распределенным реестром, где каждый из участников хранит все записи, которые считаются валидными для сети [13]. Каждый участник сети может предложить транзакцию на запись. Если определенные участники, которые заданы в протоколе консенсуса, подтверждают транзакцию, то транзакция считается валидной и все участники сети ее принимают.

В блокчейн-сетях присутствуют разные протоколы консенсуса: доказательство работы; доказательство доли; доказательство веса и др. [13]. Условие консенсуса может задаваться в виде логической формулы. Значение каждой переменной формулы представляет ответ определенного участника консенсуса. В HLF условие соглашения по транзакции, которое называют правилом одобрения, можно задать логической формулой [14]. Каждого участника правила одобрения в HLF принято называть организацией. Далее по тексту будем придерживаться данной терминологии.

Из предшествующих по времени данных можно получить статистику ответов подтверждений и отказа транзакций для каждой организации. Из собранной статистики для каждого участника можно определить вероятность подтверждения следующей транзакции с помощью байесовского подхода [15, 16]. Таким образом, достижение консенсуса является вероятностным событием.

Статистическая проверка моделей. Проверка моделей является одним из формальных методов верификации. Проверка моделей состоит из перечисленных далее трех фаз [6].

- Фаза моделирования. Моделирование системы, формализация свойств для их проверки.
- Фаза запуска, а именно запуск инструментальных механизмов проверки моделей. Проверка того, что модель удовлетворяет спецификации.
- Фаза анализа, на которой можно проверять следующие далее свойства, если модель удовлетворяет спецификации. В противном случае нужно изменить модель или спецификацию и повторить процедуру проверки модели.

В классической проверке модели не учитываются вероятностные свойства модели. Метод статистической проверки моделей позволяет проверить на модели вероятностные свойства. Метод статистической проверки моделей позволяет ответить на следующие вопросы [17]:

- количественный — вероятность того, что система удовлетворит спецификации на уровне больше порогового значения;
- качественный, показывающий какова вероятность удовлетворить требованиям спецификации.

Для ответа на поставленные вопросы используется один из перечисленных далее методов.

- Метод проверки статистических гипотез. Допустим, $p = Pr(x)$ — вероятность случайного события x. Чтобы определить, что $p \geqslant Th$, где Th — некоторое пороговое значение, необходимо протестировать гипотезу H: $p \geqslant Th$ и гипотезу K: $p < Th$.
- Метод интервальной оценки статистических параметров. Используется для определения значений случайных событий с заданным распределением.

В настоящей работе представлен ответ на качественный вопрос. При этом используется метод проверки статистических гипотез, так как метод оценки статистических параметров больше подходит для эмпирических экспериментов [18]. В методе интервальной оценки статистических параметров необходимо строить оценку вероятностных величин на основе функции распределения вероятности. Из статистических данных подтверждения и отклонения транзакции не всегда можно определить функцию распределения вероятности, поэтому применяется метод проверки статистических гипотез. Предложенное решение использует инструментальное средство PRISM [19]. Оно позволяет построить модель консенсуса в виде модели марковской цепи с дискретным временем (МЦДВ, или

discrete-time Markov chain — DTMC), а также задать спецификацию в виде pLTL-формулы.

Построение и использование модели и спецификации для протоколов многостороннего соглашения

Построение модели протокола консенсуса. Пользователь может передать конфигурацию консенсуса в следующем виде:

- множество организаций;
- для каждой организации i вероятность подтверждения транзакции P_i;
- спецификация консенсуса в виде логической формулы, которая изначально может быть задана в любом виде, однако для построения модели необходимо будет задавать формулу в дизъюнктивной нормальной форме (ДНФ).

Алгоритм 1 (см. далее) описывает создание модели консенсуса. Алгоритму передается множество пар *orgs*. Каждая пара — это имя организации и вероятность подтверждения транзакции для данной организации. Модель представляет собой бинарное дерево, где каждому узлу соответствует организация. Каждый узел хранит информацию о родителе, об ответе, который дал родитель на подтверждение транзакции, а также P_i — вероятность перехода по ребру, которое соответствует подтверждению транзакции, i — порядковый номер организации. Сначала выбирается организация для корня дерева, от которого строятся два перехода. Один переход соответствует подтверждающему ответу организации, которой поставлен в соответствие корень, второй — отказу. Переход по первому ребру происходит с вероятностью P_i , по второму — с вероятностью $1 - P_i$. Далее корень удаляется из множества *orgs* и рекурсивно строится левое и правое поддерево. Порядок организаций в множестве *orgs* не важен, так как предполагается, что вероятности подтверждений разных организаций независимы.

Построение спецификации для протокола консенсуса. Из спецификации в виде формулы ДНФ можно получить множество узлов, которые соответствуют достижению консенсуса. Алгоритм 2 (см. далее) описывает процесс получение pLTL-формулы. Для каждого литерала из изначальной спецификации в виде ДНФ происходит обход дерева марковской цепи вглубь. Если организация, которая соответствует текущему узлу обхода, находится в литерале, то совершается переход по ребру, которое соответствует подтверждающему ответу. Если же организация не присутствует в литерале, происходит переход и в левое, и в правое поддерево, так как это соответствует как подтверждению транзакции, так и отклонению ее. Далее продолжается обход вглубь по двум поддеревьям. В итоге будет получено множество листьев, каждое из которых соответствует подтверждению транзакции.

Для подтверждения транзакции необходимо достичь хотя бы один из таких листьев, поэтому строится их дизъюнкция. Так как достаточно посетить лист один раз, перед дизъюнкцией ставится временной оператор "Eventually" *F,* а также вероятностный оператор *P*.

Число узлов в модели, построенной в соответствии с алгоритмом 1, растет экспоненциально относительно числа организаций. Таким образом, алгоритм 1 имеет экспоненциальную сложность. Алгоритм 2 также имеет экспоненциальную сложность, так как в худшем случае надо обойти все поддеревья, чтобы получить листья, которые соответствуют подтверждению.

Алгоритм 1: создание модели DTMC-creation

Входные данные: множество пар <организация, вероятность> *orgs,* корень *root,* множество узлов *nodes*
Результат: Множество с узлами модели *nodes,* хранящих информацию о модели
если лист *orgs* не пустой, то:
извлечь из листа организацию *nextOrg* с вероятностью $P_{nextOrg}$
// построить поддерево с подтверждающим ответом:
создать узел $N_{nextOrg}$, родителем которого является *root*, с параметром $P_{nextOrg}$ и подтверждающим ответом;
добавить узел $N_{nextOrg}$ с подтверждающим ответом родителя в множество *nodes*;
//запустить алгоритм рекурсивно для построения поддерева:
DTMC-creation(copy(orgs), $N_{nextOrg}$*, nodes)*;
//построить поддерево с ответом отказа:
создать узел $N_{nextOrg}$, родителем которого является *root*, с параметром $N_{nextOrg}$ и с ответом отказа;
добавить узел $N_{nextOrg}$ с ответом отказа родителя в множество *nodes*;
// запустить алгоритм рекурсивно для построения поддерева:
DTMC-creation(orgs, $N_{nextOrg}$*, nodes)*;
если лист *orgs* **пустой, то:**
создать два листовых узла из корня *root* с двумя переходами, соответствующими ответам подтверждения и отказа;
добавить листья в множество *nodes*;
завершение алгоритма.

Алгоритм 2: создание спецификации для модели DTMC

Входные данные: Множество узлов модели *nodes*, спецификация в виде ДНФ *s*
Результат: спецификация в виде pLTL *s_new*
инициализировать множество листовых узлов *leafs*;
Для каждого литерала *lit* из спецификации *s* обойти дерево вглубь:
если организация, соответствующая узлу, встречается в *lit*, то совершить переход, соответствующий подтверждению транзакции;
если не встречается, то продолжить переход по двум поддеревьям: одно соответствует подтверждению, второе — отклонению транзакции;
каждый лист, который был достигнут при обходе вглубь, добавить в множество *leafs*;
s_new = дизъюнкция узлов из множества *leafs*;
добавить в начало *s_new* временной оператор *F*;
добавить в начало *s_new* вероятностный оператор *P*;
возвратить формулу *s_new*.

Использование модели и спецификации для отправки транзакции на подтверждение. После того как пользователь построил модель, можно отправлять сообщения в соответствии с этой моделью. Отправка сообщений начинается с корня дерева. Если получен ответ подтверждения, то совершается переход, соответствующий утвердительному ответу. Переход по дереву заканчивается, как только достигнут один из листьев. Если листовой узел принадлежит множеству *s_new*, которое было получено из алгоритма 2, то консенсус заканчивается принятием транзакции. При каждом переходе отправляется сообщение на подтверждение.

В некоторых моделях в случае получения ответа об отклонении транзакции может быть совершен обратный переход. Обратный переход ведет к корню дерева и инициирует процесс обхода модели заново от корня. Обратные переходы увеличивают вероятность подтверждения транзакции, но также увеличивают и число сообщений, прошедших в сети. Может быть наложено ограничение на максимальное число сообщений в модели. Более детально рассмотрим применение обратных переходов в разделе описания инструментального средства.

Инструментальные средства автоматической верификации и планирования отправки сообщений в протоколах многостороннего соглашения

В предыдущем разделе описаны алгоритмы получения модели в виде МЦДВ, а спецификации — в виде pLTL-формулы. Описан процесс отправки транзакции на подтверждение в соответствии с выбранной моделью. Таким образом, реализованы алгоритмы в инструментальные средства, которые состоят из двух описанных далее модулей.

- Consensus analyzer. Данный модуль принимает на вход конфигурацию консенсуса. В конфигурации содержится множество организаций, вероятность подтверждения транзакции для каждой организации, а также логическая формула. После этого строится модель марковской цепи и вычисляется вероятность достижения консенсуса и математическое ожидание числа сообщений. Опционально могут быть построены модели для всех возможных комбинаций обратных переходов.
- Consensus scheduler. Модуль принимает на вход модель, которая была построена модулем анализатора, а также реализацию логики отправки транзакции на подтверждение. В соответствии с моделью модуль осуществляет отправку сообщений организациям для подтверждения транзакции.

Разработанный авторами программный механизм реализован на языке Java 11. Программный код модулей можно найти в git-репозитории [20, 21]. В модуле анализатора присутствует зависимость с инструментальным средством статистической проверки моделей PRISM [19]. Модуль планировщика не имеет внешних зависимостей. На момент написания статьи модуль отправки сообщений реализован на Java, но данный модуль может быть реализован на любом языке программирования, так как он не требует внешних зависимостей. Это обеспечивает кроссплатформенность работы инструментального средства. Модель может быть построена на одной машине, где требуется Java и зависимость PRISM. Далее модель в виде текстового файла в формате .json передается на другую машину, которая подключена к блокчейн-сети. На этой машине установлен модуль отправки сообщений. В текущем разделе детально рассмотрим работу каждого из представленных выше модулей.

Consensus analyzer. Этот модуль принимает на вход конфигурацию консенсуса в формате .yaml. Конфигурация содержит имена организаций, а также вероятность принятия транзакции для каждой организации. Конфигурация содержит условие консенсуса: логическую формулу, где каждая переменная — это имя организации, участвующей в консенсусе. Для запуска модуля необходимо сохранить его исходный код и PRISM в одну и ту же директорию. Далее нужно запустить make-файл для модуля и для PRISM. После этого исходный код будет скомпилирован и можно запускать анализатор. Запускать его можно в двух режимах: первый строит одну модель без обратных переходов; второй — множество моделей со всеми уникальными комбинациями обратных переходов от узла к корню в случае, если был дан ответ

отклонения транзакции. Для каждой модели вычисляются вероятность принятия транзакции, а также математическое ожидание числа сообщений, прошедших в сети до принятия транзакции.

Модуль работает следующим образом. Сначала считывается конфигурация, а логическая формула приводится в вид ДНФ. На основе алгоритма 1 строится модель марковской цепи, а на основе алгоритма 2 — pLTL-формула. Если модуль запущен в режиме обратных переходов, то также вычисляются все возможные комбинации обратных переходов и для каждой строится модель марковской цепи и спецификация. Для каждой модели с помощью инструментальных средств PRISM вычисляется вероятность достижения консенсуса, а также математическое ожидание числа переданных сообщений до того как транзакция принята необходимым числом участников консенсуса. Результатом работы модуля является файл в формате .json, где каждой модели ставятся в соответствие вероятность и число сообщений. Модуль строит текстовое представление каждой модели, а также график зависимости вероятности подтверждения от математического ожидания числа сообщений. По графику пользователь может определить модель, которая ему наиболее подходит. Выбранную модель использует второй модуль, предназначенный для отправки сообщений.

Consensus scheduler. Данный модуль принимает на вход json-файл, который получается на выходе модуля consensus analyzer. После этого пользователю предлагается выбрать параметры алгоритма отправки сообщений, а также модель для отправки. Параметрами алгоритма отправки являются максимальное число сообщений в сети, максимальное число сообщений, которое может быть отправлено одной организации, а также время ожидания между отправками сообщений одному и тому же узлу. Модуль работает исходя из предположения, что время между отправками сообщений для одной и той же организации достаточно для того, чтобы организация изменила ответ на принятие транзакции. Порог для максимального числа сообщений может помочь в случае, если одна или несколько организаций по техническим причинам перестали отвечать. Предполагается, что достижение таймаута ответа для организации равносильно отклонению транзакции.

Модуль можно подключить как стороннюю Java-зависимость. Возможна интеграция модуля с фреймворком HLF, который имеет программный интерфейс Java, позволяющий задавать логику отправки транзакции на подтверждение. В программный код, где используется фреймворк HLF, можно подключить зависимость модуля планировщика, который примет на вход файл анализатора. Далее, используя программный интерфейс планировщика, пользователь может инициировать автоматическую отправку транзакции на подтверждение в соответствии с выбранной моделью и параметрами отправки. В следующем разделе рассмотрим ряд экспериментов с участием разработанного инструментального средства.

Экспериментальная оценка работы инструментального средства

Проиллюстрируем работу алгоритмов и программного механизма на примере построения модели консенсуса. Модель похожа на ту, которая использовалась в предыдущей работе авторов [12] по этой тематике. Однако в данном эксперименте модель строится не для взвешенного консенсуса, а для консенсуса, условие которого представлено в виде логической формулы. Эксперимент преследует следующие цели. Во-первых, следует проиллюстрировать работу предложенного программного комплекса и инструментальных механизмов статистической проверки моделей PRISM для анализа протоколов консенсуса. Во-вторых, необходимо проверить корректность работы инструментального средства и алгоритмов. В этих целях изменим параметры консенсуса и проверим, как меняется результат, который показывает программный механизм. Рассмотрим консенсус, в котором участвуют три организации с соответствующими вероятностями подтверждения транзакции, которые применялись в работе [12]:

- организация 1, имя *Org1*, вероятность подтверждения транзакции 0,93;
- организация 2, имя *Org2*, вероятность подтверждения транзакции 0,99;
- организация 3, имя *Org,* вероятность подтверждения транзакции 0,98.

Условие консенсуса выглядит следующим образом: транзакция принимается сетью, если большинство организаций, участвующих в правиле одобрения, подтвердили транзакцию. В виде логической формулы это условие можно записать следующим образом:

(*Org1* AND *Org2*) OR (*Org2* AND *Org3*)
OR (*Org1* AND *Org3*).

Модель, построенная для трех организаций по алгоритму 1, представлена на рис. 1. Разберем подробно процесс построения. Сначала из множества организаций выбираем первую организацию, которую поставим в корень, это *Org1.* Далее, строим от нее два ребра: одно соответствует подтверждению транзакции; второе — отклонению. Соответственно, переход по ребру, которое соответствует подтверждению, случается с вероятностью 0,93, а по второму ребру, которое соответствует отклонению, с вероятностью отклонению 0,07. Далее выбираем вторую организацию из множества, это *Org2*, и рекурсивно применяем алгоритм 1 к левому и правому поддеревьям с новым корнем *Org2*. Потом аналогично строим поддеревья для организации *Org3,* и от каждого узла, который соответствует *Org3,* строим два листовых узла, так как это последняя организация из множества. Модель марковской цепи для алгоритма консенсуса построена, далее рассмотрим построение спецификации для модели.

В соответствии с алгоритмом 2 для построения спецификации проходим дерево марковской цепи

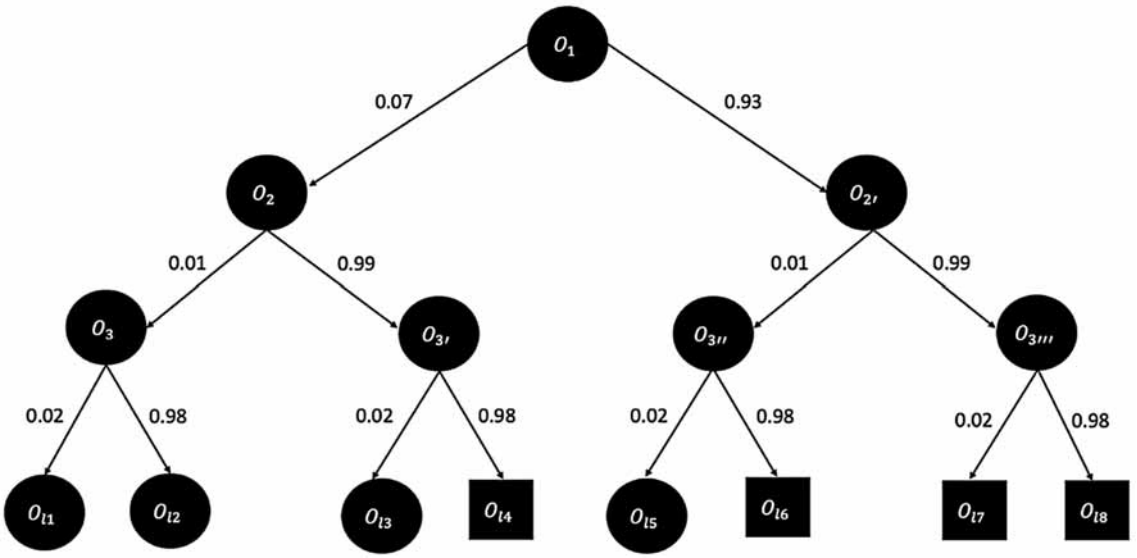

Рис. 1. Модель Марковской цепи для протокола консенсуса

вглубь для каждого литерала. Рассмотрим на примере литерала (*Org1* AND *Org2*). От узла O_1, который соответствует организации *Org1*, переходим по правому ребру, так как *Org1* присутствует в литерале, а правое ребро соответствует принятию транзакции. Аналогично для узла O_2, также переходим по правому ребру. Организация *Org3* не присутствует в литерале, поэтому совершаем переход как по правому, так и по левому ребрам. В итоге получаются узлы O_{l7}, O_{l8}, которые соответствуют принятию транзакции.

Были рассмотрены узлы только для первого литерала. Аналогично можно получить узлы, которые соответствуют принятию транзакции для второго и третьего литералов. На рис. 1 эти узлы обозначены квадратами. Для достижения соглашения по принятию транзакции необходимо посетить хотя бы один из узлов, поэтому строим дизъюнкцию узлов, которые соответствуют принятию транзакции. Так как необходимо посетить узел один раз, то ставим перед дизъюнкцией временной оператор "Eventually". В соответствии с алгоритмом 2 спецификация для марковской цепи выглядит так:

$$P = ?[F(O_{l4} \text{ OR } O_{l6} \text{ OR } O_{l7} \text{ OR } O_{l8})].$$

Таким образом, детально разобрано построение модели и спецификации, которые автоматически строятся с помощью программного модуля consensus analyzer. Далее запустим инструментальные средства и получим вероятность принятия транзакции. Спецификацию протокола консенсуса можно найти в git-репозитории [20].

Сначала запускаем анализатор без обратных переходов. Вероятность подтверждения транзакции равна 0,997728, среднее число сообщений до достижения консенсуса — 3. Этот результат соответствует ожиданиям. Так как вероятность подтверждения каждой организации по отдельности высока, то и итоговая вероятность принятия транзакции будет высокая. Принимаем во внимание всего один раунд подтверждения и три организации — ожидаемое число сообщений также равно трем.

Далее запускаем алгоритм анализа с обратными переходами. Одна из моделей с обратными переходами проиллюстрирована на рис. 2.

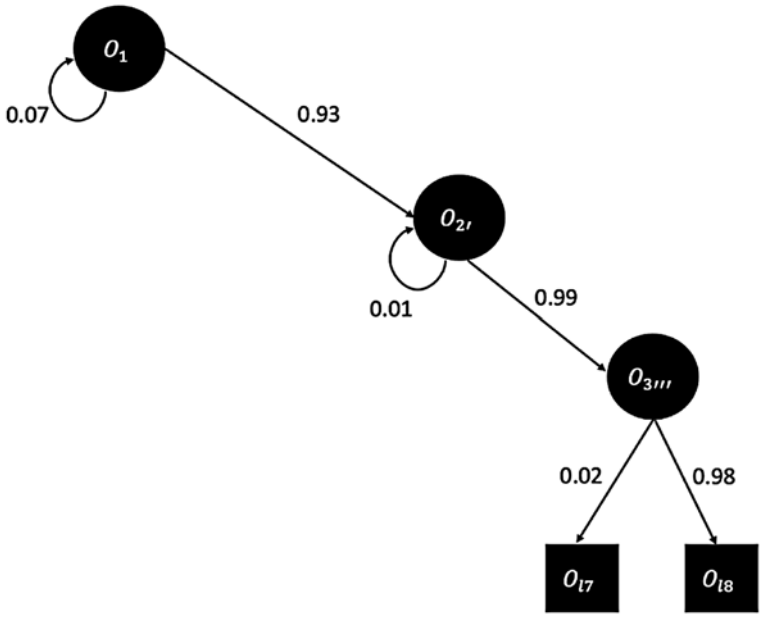

Рис. 2. Модель Марковской цепи с обратными переходами

Выбрана модель с обратными переходами, которая соответствует наибольшей вероятности достижения консенсуса при наименьшем числе сообщений. Вероятность принятия транзакции равна 1, математическое ожидание числа сообщений 3,2212. Как видно на рис. 2, все переходы ведут к листьям, которые соответствуют принятию транзакции O_{l7}, O_{l8}, что дает стопроцентную вероятность подтверждения транзакции.

Теперь рассмотрим, как меняется результат анализа при изменении параметров консенсуса. Поднимем вероятность подтверждения транзакции для первой и второй организаций до 1. Программный механизм показывает, что вероятность принятия транзакции в этом случае также равна 1. Это ожидаемо, так как в спецификации вероятность, что первый литерал будет верен, равна 1.

Проведем противоположный эксперимент: снизим вероятность принятия транзакции для организации 3 до нуля. В этом случае анализатор показывает, что вероятность принятия транзакции равна 0,9207. Это также соответствует ожиданиям, так как это число равно произведению вероятностей первой и второй организаций: $0{,}93 \times 0{,}99 = 0{,}9207$.

Итак, показано, что результаты экспериментов, полученные в ходе работы инструментального средства, совпадают с теоретическими расчетами. Далее одну из моделей можно передать на вход модулю consensus scheduler. Модуль consensus scheduler позволяет выбрать модель с наименьшим математическим ожиданием числа сообщений в сети, или модель с наивысшей вероятностью подтверждения, либо любую другую модель, построенную модулем consensus analyzer. Выбор модели остается на усмотрение пользователя.

Заключение

В настоящей работе представлены разработанные авторами алгоритмы построения модели и спецификации для протоколов консенсуса, задаваемой в виде логических формул. Продемонстрировано, как на основе выбранной модели пользователь может автоматически отправлять транзакции на подтверждение. Разработанный программный механизм реализует предложенные алгоритмы. Он состоит из двух программных модулей. Модуль для анализа консенсуса consensus analyzer определяет вероятность принятия транзакции, а также математическое ожидание числа сообщений, прошедших в сети до подтверждения транзакции. После этого результат анализатора может быть передан модулю consensus scheduler, который инициирует отправку сообщений в соответствии с выбранной моделью. Разработанный программный механизм может быть интегрирован с фреймворком Hyperledger Fabric. Перспективным направлением является его использование в промышленных системах, а также проведение испытаний с помощью метода ab-test [22] с целью выявить изменение числа подтверждения транзакций с работой инструментальных средств и без них.

Список литературы

1. **Jaoude J. A., Saade R. G.** Blockchain Applications — Usage in Different Domains // IEEE Access. 2019. Vol. 7. P. 45360—45381.

2. **Bach L., Branko M., Zagar M.** Comparative analysis of blockchain consensus algorithms // MIPRO. 2018. P. 1545—1550.

3. **Androulaki E., Barger A., Bortnikov V., Cachin C., Christidis K., De Caro A., Enyeart D., Ferris C., Laventman G., Manevich Y.** Hyperledger fabric: a distributed operating system for permissioned blockchains // Proceedings of the Thirteenth EuroSys Conference. 2018. P. 1—15.

4. **Saad M., Spaulding J., Njilla L., Kamhoua C., Shetty S., Nyang D., Mohaisen A.** Exploring the attack surface of blockchain: A systematic overview // IEEE Communications Surveys & Tutorials. 2020. Vol. 22, No. 3. P. 1977—2008.

5. **Dabholkar A., Saraswat V.** Ripping the fabric: Attacks and mitigations on hyperledger fabric // Applications and Techniques in Information Security. ATIS 2019. Communications in Computer and Information Science. 2019. Vol. 1116. P. 300—311.

6. **Baier C., Katoen J.** Principles of model checking. MIT press, 2008. 984 p.

7. **Legay A., Delahaye B., Bensalem S.** Statistical model checking: An overview // Runtime Verification. RV 2010. Lecture Notes in Computer Science. 2010. vol. 6418. P. 122—135.

8. **Федотов И. А., Хританков А. С.** Систематический обзор исследований в области автоматической верификации кода смарт-контрактов // Программная инженерия. 2020. Том 11, № 1. С. 3—13.

9. **Agha G., Palmskog K.** A survey of statistical model checking // *ACM Transactions on Modeling and Computer Simulation (TOMACS)*. 2018. Vol. 28, No. 1. P. 1—39.

10. **Fedotov I., Khritankov A.** Statistical Model Checking of Common Attack Scenarios on Blockchain // Proceedings of the 9th International Symposium on Symbolic Computation in Software Science. 2021. P. 65—77.

11. **Imeri A., Agoulmine N., Khadraoui D.** Smart Contract modeling and verification techniques: A survey // 8th International Workshop on ADVANCEs in ICT Infrastructures and Services. 2020. P. 1—8.

12. **Fedotov I., Khritankov A., Barger A.** Towards automated verification of multi-party consensus protocols. arXiv preprint. 2021.

13. **Zheng Z., Xie S., Dai H., Chen X., Wang H.** An overview of blockchain technology: Architecture, consensus, and future trends // 2017 IEEE International Congress on Big Data. 2017. P. 557—564.

14. **Cachin C.** Architecture of the hyperledger blockchain fabric // Workshop on distributed cryptocurrencies and consensus ledgers. 2016. Vol. 310, No. 4. P. 1—4.

15. **Duda J.** Exploiting statistical dependencies of time series with hierarchical correlation reconstruction// CoRR. 2018. P. 11—24.

16. **Corani G., Benavoli A.** A bayesian approach for comparing cross-validated algorithms on multiple data sets // Machine Learning. 2015. Vol. 100, No. 2-3. P. 285—304.

17. **Legay A.** Statistical model checking // Computing and Software Science. Lecture Notes in Computer Science. 2016. Vol. 10000. P. 478—504.

18. **Younes H., Simmons R.** Probabilistic verification of discrete event systems using acceptance sampling // Computer Aided Verification. CAV 2002. Lecture Notes in Computer Science. 2002. Vol. 2404. P. 223—235.

19. **Kwiatkowska M., Norman G., Parker D.** PRISM 4.0: Verification of probabilistic real-time systems // Computer Aided Verification. CAV 2011. Lecture Notes in Computer Science, 2011. Vol. 6806. P. 585—591.

20. **Fedotov I., Morounfoluwa D. O.** Consensus Analyzer. URL: https://github.com/1vanan/consensus-analyzer

21. **Fedotov I., Morounfoluwa D. O.** Consensus Scheduler. URL: https://github.com/1vanan/consensus-scheduler

22. **Gronau Q., Akash Raj N., Wagenmakers E.** Informed Bayesian Inference for the A/B Test. arXiv preprint. 2019.

Automated Verification of Multi-Party Agreements and Scheduling of Sending Messages in Distributed Ledger Systems

I. A. Fedotov[1], ivan.fedotov@phystech.edu, **A. S. Khritankov**[2], anton.khritankov@acm.org,
M. D. Obidare[1], obidarefolu1@gmail.com,
[1] MIPT, Dolgoprudny, Moscow Region, 141701, Russian Federation,
[2]HSE, Moscow, 115432, Russian Federation

Corresponding author:

Fedotov Ivan A., Postgraduate Student, MIPT, Dolgoprudny, Moscow Region, 141701, Russian Federation
E-mail: ivan.fedotov@phystech.edu

Received on February 28, 2022
Accepted on March 06, 2022

One can use a multi-party agreement in distributed ledger systems and blockchain networks to reach an agreement on changes of the state of the system. If one of the network members proposes a transaction, then certain network participants shall confirm it. After that the whole network can consider transaction as a valid one. A multi-party agreement or consensus determines the composition of these participants. Based on the historical data set, one can calculate the probability of confirming a transaction for each of the participants. In this paper, we use a statistical model checking approach to determine the likelihood that the network accepts a transaction. Sending confirmation requests may require an additional fee. We calculate the probability, and the mathematical expectation of the number of messages before reaching a consensus. Further, consensus models are built in the form of a Markov chain with various strategies for sending messages. Based on the proposed methods, we design a tool that automatically builds models for various strategies of sending messages and verifies the model using a statistical model verification approach. After choosing the optimal model, one can send confirmation messages using the scheduler module of developed tool.

Keywords: *consensus, blockchain, verification, statistical model checking*

For citation:

Fedotov I. A., **Khritankov A. S.**, **Obidare M. D**. Automated Verification of Multi-Party Agreements and Scheduling of Sending Messages in Distributed Ledger Systems, *Programmnaya Ingeneria,* 2022, vol. 13, no. 4, pp. 200—208.

DOI: 10.17587/prin.13.200-208

References

1. **Jaoude J. A., Saade R. G.** Blockchain Applications — Usage in Different Domains, *IEEE Access,* 2019, vol. 7, pp. 45360—45381.

2. **Bach L., Branko M., Zagar M.** Comparative analysis of blockchain consensus algorithms, *MIPRO*, 2018, pp. 1545—1550.

3. **Androulaki E., Barger A., Bortnikov V., Cachin C., Christidis K., De Caro A., Enyeart D., Ferris C., Laventman G., Manevich Y.** Hyperledger fabric: a distributed operating system for permissioned blockchains, *Proceedings of the Thirteenth EuroSys Conference,* 2018, pp. 1—15.

4. **Saad M., Spaulding J., Njilla L., Kamhoua C., Shetty S., Nyang D., Mohaisen A.** Exploring the attack surface of blockchain: A systematic overview, *IEEE Communications Surveys & Tutorials*, 2020, vol. 22, no. 3, pp. 1977—2008.

5. **Dabholkar A., Saraswat V.** Ripping the fabric: Attacks and mitigations on hyperledger fabric, *Applications and Techniques in*

Information Security. ATIS 2019. Communications in Computer and Information Science, 2019, vol. 1116, pp. 300—311.

6. **Baier C., Katoen J.** *Principles of model checking,* MIT press, 2008. 984 p.

7. **Legay A., Delahaye B., Bensalem S.** Statistical model checking: An overview, *Runtime Verification. RV 2010. Lecture Notes in Computer Science*, 2010, vol. 6418. pp. 122—135.

8. **Fedotov I. A., Khritankov A. S.** Systematic Review of Automatic Verification of Smart-Contracts, *Programmnaya Ingeneria,* 2020, vol. 11, no. 1, pp. 3—13 (in Russian).

9. **Agha G., Palmskog K.** A survey of statistical model checking, *ACM Transactions on Modeling and Computer Simulation (TOMACS),* 2018, vol. 28, no. 1, pp. 1—39.

10. **Fedotov I., Khritankov A.** Statistical Model Checking of Common Attack Scenarios on Blockchain, *Proceedings of the 9th International Symposium on Symbolic Computation in Software Science*, 2021, pp. 65—77.

11. **Imeri A., Agoulmine N., Khadraoui D.** Smart Contract modeling and verification techniques: A survey, *8th International Workshop on ADVANCEs in ICT Infrastructures and Services,* 2020, pp. 1—8.

12. **Fedotov I., Khritankov A., Barger A.** Towards automated verification of multi-party consensus protocols, arXiv preprint. 2021.

13. **Zheng Z., Xie S., Dai H., Chen X., Wang H.** An overview of blockchain technology: Architecture, consensus, and future trends, *2017 IEEE International Congress on Big Data,* 2017, pp. 557—564.

14. **Cachin C.** Architecture of the hyperledger blockchain fabric, *Workshop on distributed cryptocurrencies and consensus ledgers,* 2016, vol. 310, no. 4, pp. 1—4.

15. **Duda J.** Exploiting statistical dependencies of time series with hierarchical correlation reconstruction, *CoRR,* 2018, pp. 11—24.

16. **Corani G., Benavoli A.** A bayesian approach for comparing cross-validated algorithms on multiple data sets, *Machine Learning,* 2015, vol. 100, no. 2-3. pp. 285—304.

17. **Legay A.** Statistical model checking, *Computing and Software Science. Lecture Notes in Computer Scienc*e, 2016, vol. 10000, pp. 478—504.

18. **Younes H., Simmons R.** Probabilistic verification of discrete event systems using acceptance sampling, *Computer Aided Verification. CAV 2002. Lecture Notes in Computer Science,* 2002, vol. 2404, pp. 223—235.

19. **Kwiatkowska M., Norman G., Parker D.** PRISM 4.0: Verification of probabilistic real-time systems, *Computer Aided Verification. CAV 2011. Lecture Notes in Computer Science,* 2011, vol. 6806, pp. 585—591.

20. **Fedotov I., Morounfoluwa D. O.** Consensus Analyzer, available at: https://github.com/1vanan/consensus-analyzer

21. **Fedotov I., Morounfoluwa D. O.** Consensus Schefuler, available at: https://github.com/1vanan/consensus-scheduler

22. **Gronau Q., Akash Raj N., Wagenmakers E.** Informed Bayesian Inference for the A/B Test. arXiv preprint. 2019.

ООО "Издательство "Новые технологии". 107076, Москва, ул. Матросская Тишина, д. 23, стр. 2
Технический редактор *Е. М. Патрушева*. Корректор *А. В. Чугунова*.

Сдано в набор 09.03.2022 г. Подписано в печать 05.04.2022 г. Формат 60×88 1/8. Заказ PI421
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье **K. I. Gaydamaka, A. D. Belonogova**
«APPLYING UNSUPERVISED MACHINE LEARNING ALGORITHMS TO ENSURE REQUIREMENTS CONSISTENCY»

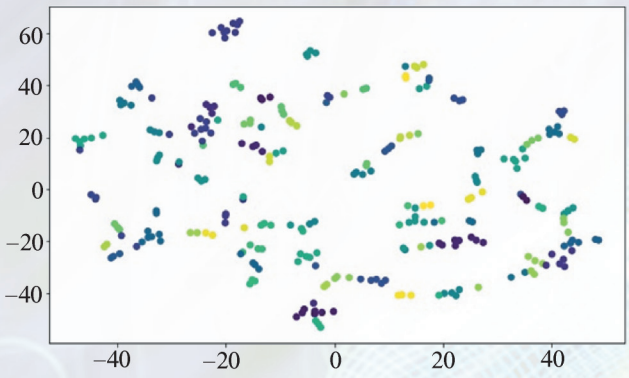

Fig. 6. K-means clustering

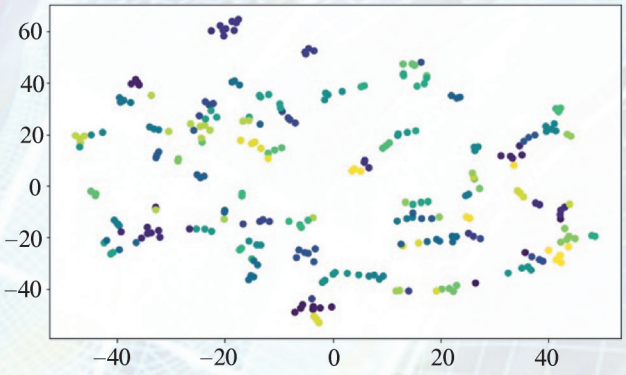

Fig. 8. EM clustering

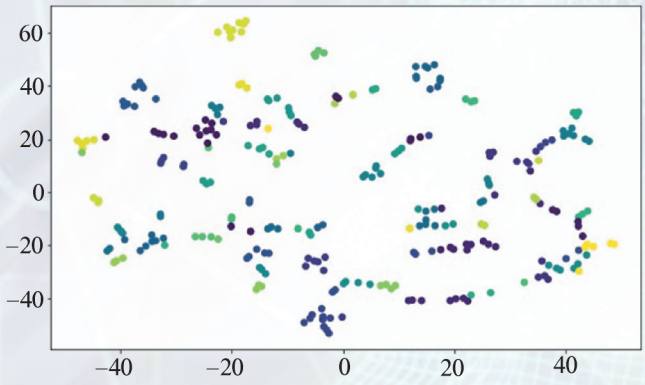

Fig. 10. Clustering Agglomerative Average

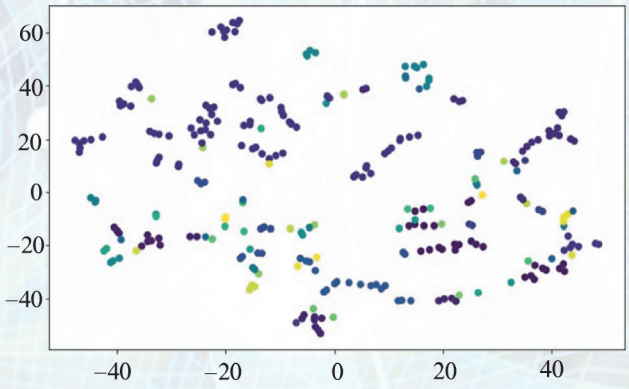

Fig. 12. Clustering Agglomerative Single

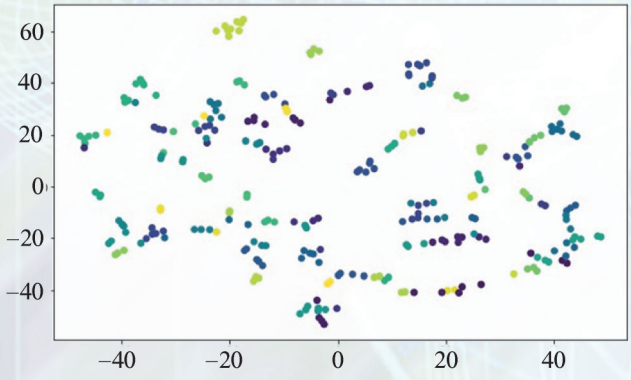

Fig.13. Clustering Agglomerative Complete

Рисунки к статье **K. I. Gaydamaka, A. D. Belonogova**
«APPLYING UNSUPERVISED MACHINE LEARNING ALGORITHMS TO ENSURE REQUIREMENTS CONSISTENCY»

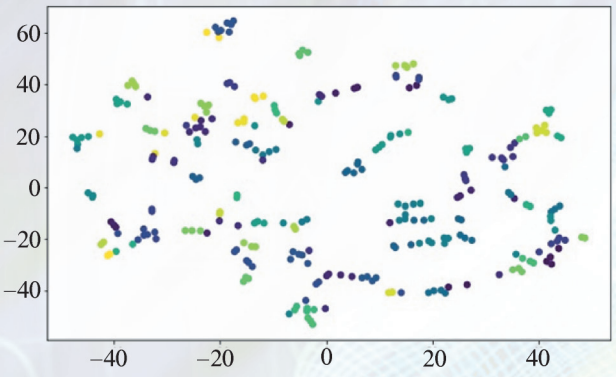

Fig. 15. Clustering Agglomerative Ward

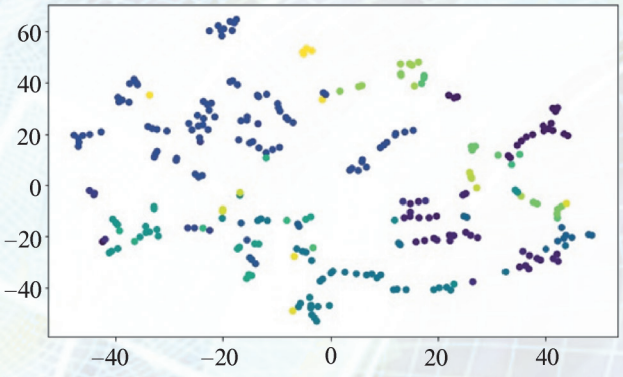

Fig. 17. DBSCAN clustering

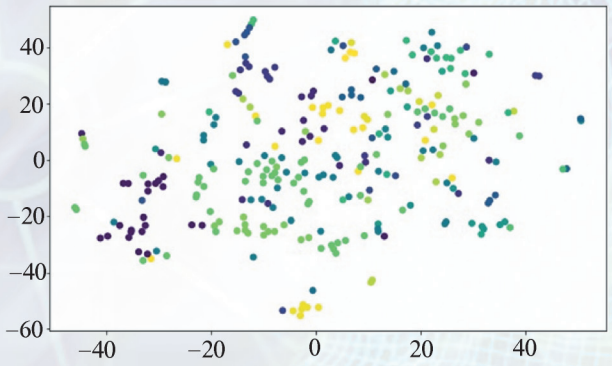

Fig. 18. Clustering the method of simple cosine distances

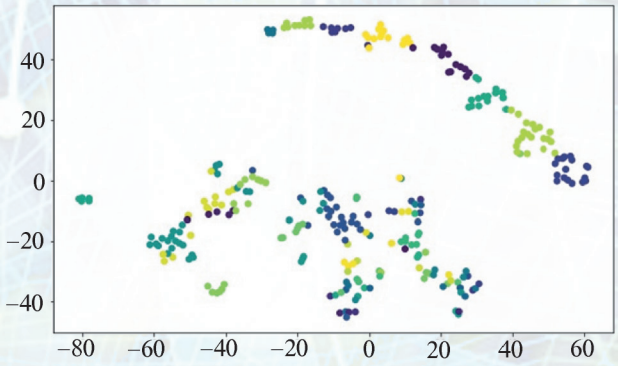

Fig. 19. Doc2Vec DBOW clustering

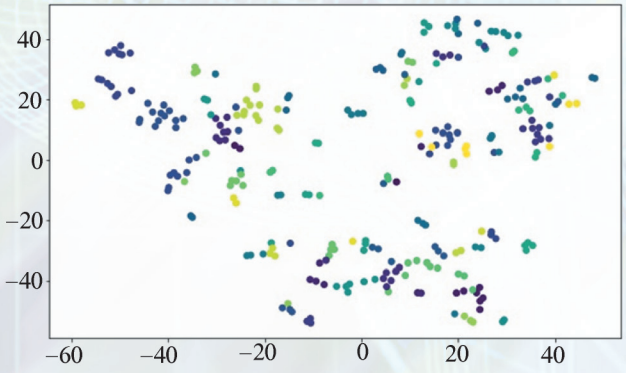

Fig. 21. BERT clustering

ISSN 2220-3397. Программная инженерия. 2022. Том 13. № 4. 153 – 208. Подписной индекс по Объединенному каталогу "Пресса России" – 22765