УДК 004.021
DOI: 10.46548/21vek-2020-0951-0016

СОЗДАНИЕ КОРПОРАТИВНОГО ТЕКСТОВОГО РЕДАКТОРА ПРИ ПОМОЩИ РЕПЛИЦИРУЕМЫХ НЕ КОНФЛИКТНЫХ ТИПОВ ДАННЫХ

Саватеев Максим Валерьевич, магистрант кафедры «Вычислительные машины и системы»
Калашников Виталий Александрович, магистрант кафедры «Вычислительные машины и системы»
Мартышкин Алексей Иванович, кандидат технических наук,
доцент кафедры «Вычислительные машины и системы»
Пензенский государственный технологический университет
(440039, Россия, Пенза, проезд Байдукова/ул. Гагарина, д. 1а/11,
e-mails: never-ok2014@yandex.ru, mymailfordev@gmail.com, alexey314@yandex.ru)
Гурин Евгений Иванович, доктор технических наук,
профессор, профессор кафедры «Вычислительная техника»
Пензенский государственный университет
(440026, Россия, Пенза, улица Красная, 40, e-mail: gurin2@yandex.ru)

Аннотация. В данной статье рассматриваются основные трудности, связанные с реализацией приложений для совместного редактирования документов как распределенной системы. Перечисляются самые распространенные методы реализации алгоритмов для создания коорпаративного редактора текста, такие как блокирования редактируемого сегмента текста при редактировании одним соавтором; дифференциальное трехстороннее слияние изменений между клиентами и сервером; использование древовидных структур данных; алгоритмы на основе реплицируемых не конфликтных типов данных с подходом операционных преобразований. Приводятся основные достоинства и недостатки перечисленных методов. Предложен вариант реализации с использованием алгоритмов, основанных на реплицируемых не конфликтных типов данных с подходом операционных преобразований. Рассматривается пути решения основной проблемы расхода памяти, характерной для данного метода.

Ключевые слова: клиент, сервер, текстовый редактор, CRDT, распределенные системы.

CREATING CORPORATIVE TEXT EDITOR BY USING CONFLICT-FREE REPLICATED DATA TYPES

Savateev Maxim Valerievich, student of sub-department «Computers and systems»
Kalashnikov Vitaliy Alexandrovich, student of sub-department «Computers and systems»
Martyshkin Alexey Ivanovich, candidate of technical sciences, docent,
associate professor of sub-department «Computers and systems»
Penza state technological University
(440039, Russia, Penza, Baydukov Proyezd / Gagarin Street, 1a/11,
e-mails: never-ok2014@yandex.ru, mymailfordev@gmail.com, alexey314@yandex.ru)
Gurin Evgeny Ivanovich, doctor of technical Sciences, Professor,
professor of the Department of «Computer engineering»,
Penza state University
(440026, Russia, Penza, Krasnaya street, 40, e-mail: gurin2@yandex.ru)

Abstract. This article describes test automation methodology that can cover all levels of automated testing of distributed information systems with a complex architectural organization using the theory of automata which are necessary for the visualization of this mechanism. Interpreting the execution sequence of a test scenario using a finite state machine makes it easier to understand the process of performing automated testing. This type of testing, unlike manual testing, includes various levels that ensure quality control of the product throughout its life cycle. There are many different automated testing methodologies that are designed to implement these levels. Two main groups can be distinguished from this set: a methodology based on keywords and a methodology based on the use of models in an explicit form. However, each of them has its drawbacks narrowing the area of application. The new methodology proposed in this article is universal, combining a high-level abstraction that facilitates the process of creating a test set, and in-depth verification of system specifications using a testing mechanism that allows you to improve the quality of the product being developed by fully covering test scenarios. The methodology designed in this article is relevant for systems with complex architectural organization.

Keywords: Client, server, text editor, CRDT, distributed systems.

Введение. Жизнь современного человека невозможно представить без участия в ней компьютера. Ежедневно человек взаимодействует с компьютером, удовлетворяя свои потребности. Огромную роль при взаимодействии с компьютером играют текстовые редакторы [1]. По характеру использования текстовые редакторы делят на 2 группы:

- для личного использования;
- для корпоративного использования.

Корпоративный характер использования широко

распространён в крупных корпорациях и компаниях, где доступ к одному документу необходим сразу нескольким пользователям (от 2 и более). В таких случаях нередко происходит ситуация, когда несколько пользователей одновременно хотят внести изменения в одно и то же место в документе. Подобный исход событий принято считать конфликтным.

Разрешать такие конфликты призвана так называемая «*Теорема CAP*», которая утверждает, что в любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств:

• согласованность данных (*consistency*) – во всех вычислительных узлах в один момент времени данные не противоречат друг другу;

• доступность (*availability*) – любой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;

• устойчивость к разделению (*partition tolerance*) – расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

Условно все подходы реализации облачных редакторов текста можно разделить на два класса [2]:

• с использованием блокирующих алгоритмов;

• без использования блокирующих алгоритмов [3].

Подход с использованием блокирующих алгоритмов [4] заключается в следующем: если один пользователь в определённый момент времени работает с документом, то в случае, если другой пользователь попытается внести свои изменения, у него ничего не выйдет, т. к. участок документа, с которым работает первый пользователей будет заблокирован для второго пользователя [5].

Достоинством данного подхода является простата реализации [6], требующая от разработчика только разработки протокола взаимодействия клиентов с сервером, в котором необходимо предусмотреть аварийное освобождение занятой секции в случае бездействия редактирующего соавтора [7].

К недостаткам можно отнести отсутствие возможности редактирования документа в режиме *offline* с дальнейшим применением изменений в тексте на сервере. Огромный размер передаваемой информации для редактирования сегмента документа и возврата на сервер для применения изменений. Так же невозможно редактирование документа группой лиц, что принижает роль приложения как колбаративного редактора.

Для решения проблем блокирующих алгоритмов были разработаны методы, которые можно объединить под общим названием – не блокирующие алгоритмы.

Все подобные алгоритмы подразумевают в своей реализации использование так называемых «реплик» [7]. Реплика – копия документа, отправленная со стороны клиента, над которой далее производятся изменения. То есть реплика – своего рода «слепок» документа. Алгоритмы должны гарантировать, что все реплики документов будут идентичны у всех участников в распределенной системе.

Основные методы реализации неблокирующих алгоритмов перечислены далее:

• использование алгоритмов дифференциальной синхронизации [8];

• использование «древовидных» структур данных с использованием *Zipper'*ов;

• использование алгоритмов операционных преобразований (далее *ОП*);

• использование неконфликтных реплицируемых типов данных (далее *CRDT*).

Материалы и результаты исследований. Для решения проблем, присутствующих в блокирующем методе, был разработан ряд решений без использования блокировок. Метод основывается на постоянном слиянии изменений на стороне клиента и на стороне сервера с рассылкой изменений в обе стороны.

Сам процесс назван трехсторонним слияниям и включает три основных шага:

1) клиент отправляет документ на сервер;

2) сервер собирает все изменения клиента и других участников и применяет их к документу;

3) сервер рассылает копии документа всем клиентам.

Данный подход позволяет вводить изменения в документ без блокирования [9]. Однако ставит перед разработчиком сложную задачу реализации механизма слияния изменений, сам же алгоритм затрачивает значительное количество времени на обработку документа. Данное обстоятельство не позволяет использовать этот метод в условиях высоконагруженной вычислительной сети. Так же к недостаткам можно отнести отсутствие гарантии, что после применения очередного слияния сложно структурированный документ не будет испорчен.

Для фиксирования изменений в документе с сохранением иммутабельности первоначального и последующих состояний документа, применяют «древовидные» структуры данных с использованием *Zipper'*ов [10].

Метод заключается в представлении документа как бинарного дерева, листьями которого являются символы в тексте. Во время редактирования для данного документа создаётся *Zipper*, который содержит измененные символы. Для создания *zipper*'а в бинарном дереве создаётся новая вершина, соответствующая корневой вершине, к ней прикрепляются ссылки на дочерние вершины из исходного дерева, ведущие к изменяемому сегменту. На следующем шаге создаётся новый лист взамен отредактированного текста [11]. На последнем шаге создается новая версия дерева, где *zipper* заменяет собой отредактированную ветвь, при этом старое дерево и сам *zipper* хранится отдельно в памяти. Достоинством данного метода является возможность использовать инструменты контроля состояния документа и легко переключаться между старыми и новыми версиями.

К основным недостаткам метода можно отнести

расход памяти, вызванный огромным размером структур данных и сохранением в памяти всех *zipper* для возможности построения старых версий документа. Так же большой размер имеет сам пакет с вектором изменений, который необходимо передать участникам.

Метод операционных преобразований (ОП). Данный метод заключается в представлении каждого события ввода или удаления символа как операции и рассылки всем участникам этой операции для применения изменений. Данный концепт позволяет соблюсти согласованность в копиях документа [12]. Были разработаны ряд моделей для поддержки согласованности, однако каждый требовал сложный алгоритм взаимодействия для предотвращения нарушения причинно-следственной связи в документе. Схема работы данного метода представлена на рисунке 1.

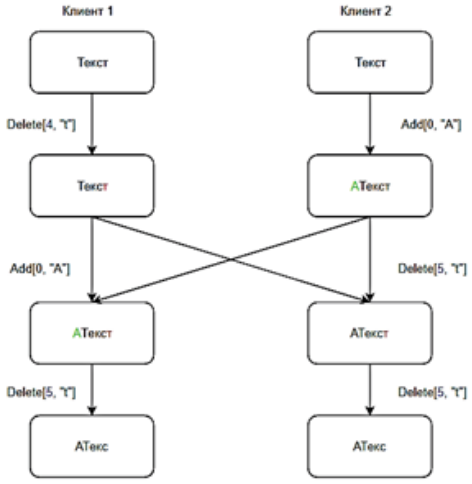

Рисунок 1 – Схема работы метода Операционных преобразований

Среди всех перечисленных ранее методик, ОП является самой сложной в реализации. К недостаткам данного метода относится вероятность возникновения события, когда результат работы метода оказывается некорректным. Это происходит, когда пользователи начинают редактировать один и тот же участок текста, используя разные операции. Все изменения, проводимые пользователями, происходят одновременно. После того, как метод попытается применить все пользовательские изменения, исходный результат не соответствует ожиданиям и является некорректным. Наглядное представления данной ситуации представлено ниже на рисунке 2.

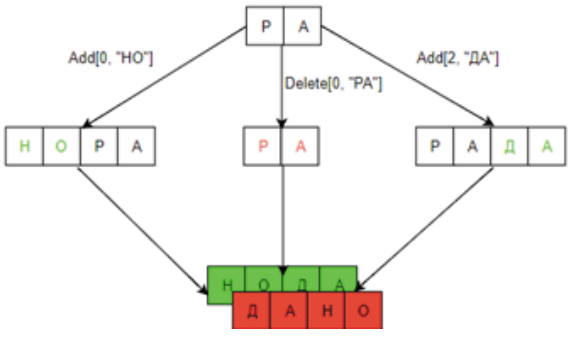

Рисунок 2 – Некорректная работа метода

Метод неконфликтных реплицируемых типов данных. Логическим продолжением *ОП* стало использование алгоритмов и типов данных *conflict-free replicated data type (CRDT)* [13]. Данный метод решает проблемы, которые присутствуют в *ОП*, а именно контроль когерентности состояний всех копий документа с поддержкой согласованности применяемых изменений во всех репликах документа. Стоит отметить, что алгоритм намного проще реализовать, чем алгоритм взаимодействия в *ОП*. При редактировании документа создается вектор *CRDT* объектов, который после прекращения редактирования отправляется на сервер, серверу в свою очередь достаточно распределить вектор в порядке возрастания версий векторов. Для рассылки изменений достаточно отправить те вектора, чья версия больше, чем версия, хранимая на клиенте. Для *CRDT* характерны несколько свойств, таких как ассоциативность, выраженная по формуле 1, и коммутативность, выраженная по формуле 2.

$$(a \cup b) \cup c = a \cup (b \cup c), \quad (1)$$

$$a \cup b = b \cup a, \quad (2)$$

Для разных задач удобнее применять разные реализации данного метода. *CRDT* включает в себя два класса: *CvRDT* и *CmRDT*. В реализации *CvRDT* (*Convergent Replicated Data Type*) всем репликам отправляется полностью вся структура данных т.е. в случае изменения списка отправляется весь список. Структура данных должна поддерживать операции:

• *query* - прочитать какой-либо элемент, не изменяя состояние.

• *update* - изменить структуру, например добавить элемент в список.

• *merge* - объединить состояние, пришедшее из другой реплики с данным состоянием. Данная операция должна быть коммутативной, ассоциативной и идемпотентной. Слияние любых состояний в любых направлениях не возвращает систему в более раннее состояние, а монотонно увеличивает состояние системы.

В реализации *CmRDT (Commutative Replicated Data Type)*, при добавлении элемента репликам отправляется только изменённое состояние, например только добавленный в список элемент, а не весь список. Операции должны быть коммутативными, чтобы состояние реплики не зависело от порядка получения апдейтов.

Достоинствами данного метода являются простота реализации, относительно метода операционных преобразований, соблюдение таких свойств, как коммутативность, ассоциативность, идемпотентность, монотонность. Благодаря перечисленным свойствам можно избежать нарушения порядка, приоритета, потерю пакетов в сети. Также достоинством метода является редактирование произвольных строк.

Недостатком данного подхода является скопления векторов изменений тех символов, которые уже не существуют в документе и не имеют информативной ценности, для этих целей необходима сборка мусора. Если не осуществить сборку мусора, то при подключе-

нии нового клиента, ему будет отправлен пакет огромного размера. Сам клиент будет вынужден затратить огромный объем времени на построение документа.

На рисунке 3а продемонстрирована схема изменений состояний клиента и хранимого документа в виде конечного автомата, где {*S1, S2, S3, S4*} множество состояний, {*O1, O2*} множество операций над документом, {*T0, T1, T2, T3*} текст документа, хранимый на клиенте. Для алгоритмов, основанных на *CRDT* и *ОП*, важно чтобы каждый участник начинал с общего начального состояния документа, обозначенный на рисунке 3 как *S1*, при применении операций *O1* и *O2* все участники должны получить единое состояние *S4* с вариантом текста *T3* не зависимо от порядка применения операций *O1* и *O2*. Если какому-либо участнику не удалось прийти к конечному общему состоянию, то его участи в коллективном редактировании может привести к ошибкам.

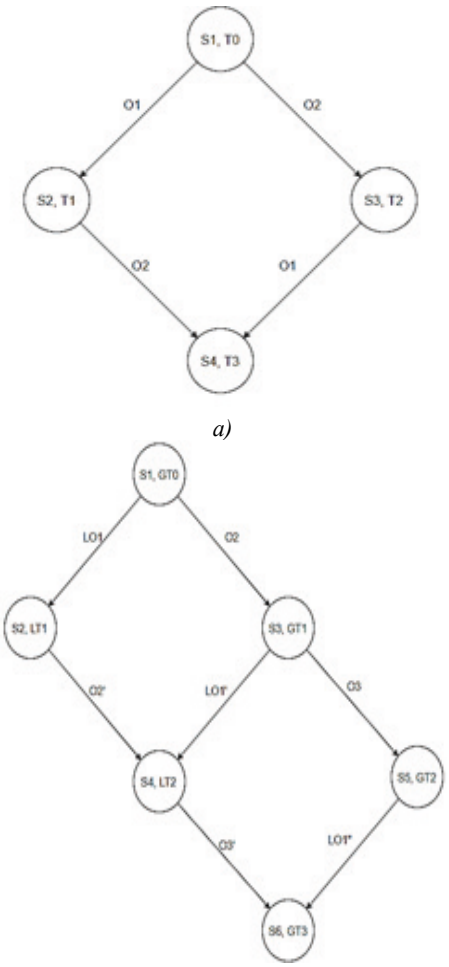

Рисунок 3 – Графическое представление состояний клиента (а) и состояний клиента относительно глобальной версии документа (б)

Для подробного разбора взаимодействия клиента с другими участниками в совместном редактировании, на рисунке 3б продемонстрирована схема изменений состояний клиента и хранимого документа относительно глобальной общей версии документа в виде конечного автомата, где {*S1,..,S6*} множество состояний; {*LO1, LO1', LO1"*} множество операций над локальной версией документа клиента; {*O1, O2, O3, O2', O3'*} множество операций над документом; {*GT0, GT1, GT2, GT3*} текст документа, совпадающий с глобальной версией, {*LT1, LT2*} текст документа, хранимый на локальном клиенте.

Множество {*GT0, GT1, GT2, GT3*} представляет ревизии глобального документа после каждого изменения. Как видно по схеме для получения итоговой ревизии *GT3* клиенту необходимо применить операцию *O3*', когда как всем участникам необходимо применить операцию *LO3*".

Ранее упоминалось множество {*LO1, LO1', LO1"*}. Данное множество образуется путем обработки операции *LO1* участниками после каждого изменения в тексте, обработка необходима для проведения изменения в нужной позиции над нужным символом. Клиент так же обрабатывает глобальные операции *O2* и *O3* при изменении локальной копии документа, результатом обработки являются операции *O2'* и *O3*'.

Реализация взаимодействия, основанного на CRDT. Стоит отметить, что использование *CRDT* требует от сервера только централизованное хранение документов и векторов изменений [14] этих самых документов, если этого не требуется, то есть возможность реализовать одноранговое *peer-to-peer* взаимодействие [15].

В методе будет рассмотрено классическое клиент-серверное взаимодействие т.к. основным критерием является хранение документа, а также возможность получить его реплику в любой момент времени [16].

CRDT представляет собой объект класса, хранящий:

- литерал, над которым применили операцию;
- код самой операции;
- индекс в тексте, где применялась операция;
- номер версии, в которой вносилось изменение;
- дата применяемых изменений, требуется для сборки мусора.

Вся эта информация позволяет обеспечить согласованность во всех репликах документа [17].

Алгоритм построения реплики документа основывается на последовательном применении операций с соблюдением монотонного возрастания счетчика версий и устранения ошибок, в случае если обнаружится отсутствие какой-либо версии. Одним из способов устранения данной ошибки является попытка запросить у участников вектор нужной версии.

Результатом последовательного применения вектора изменений от версии к версии является единая реплика документа, которая будет идентична у всех участников [18]. Однако если произойдет слияние двух и более векторов разных участников в рамках одной версии, то это приведет к нарушению когерентности документа. Для предотвращения данной ошибки стоит обрабатывать изменения клиентов в строгой очереди и разграничивать доступ к общему вектору версий.

Важной задачей является процедура сборки мусора. Возможны разные варианты реализации, основанные на разграничении прав пользователей и/или степени важности документа, однако все они основываются на том, факте, что применение взаимно противоположных операций над одним и тем же литералом не введет к прямому изменению документа. Это можно представить по формуле 3.

$$insert(ch,i,v)+delete(ch,i',v')=0, \quad (3)$$

где *ch* – литерал, над которым применяется операция;

i – индекс литерала в тексте;

i' - изменений индекс литерала в связи с применением промежуточных операций;

v и *v'* – версия, в которой применялись изменения.

Сборка мусора заключается в удалении в векторе таких пар операций, которые не вносят в документ изменений как таковых, например, удаление ранее присутствующего литерала. Однако факт присутствия литерала ранее в тексте оказывает влияния на сами промежуточные операции. Данное обстоятельство требует внести изменения в сами промежуточные операции, которые заключаются в изменении индексов. В случае замены литерала на другой достаточно удалить пару вставки-удаления старого литерала и на их место поставить операцию вставки нового литерала.

Однако данная операция нарушает иммутабельность [19] начального и промежуточного документа и не позволит получить старые версии текста. Компромиссным решением является установка ограничений на применение сборки мусора, опираясь на «возраст» изменений, конечной версии для чистки, ориентация на степень важности документа. Так же можно предоставить возможность запуска сборки мусора пользователю с определенной ролью, дав возможность настраивать критерии процедуры.

Важную роль играет разрешение конфликта при одновременном редактировании документа. В данном случае документ представляется как критический ресурс, доступ к которому осуществляется только одному участнику редактирования. Доступ к документу представляется как критическая секция. Работу с данным ресурсом можно представить в виде математической модели алгоритма, описанной на языке *СНДА*.

Механизм разрешения конфликтов представлен на рисунке 4. Ниже описаны условные обозначения пошаговой работы алгоритма разрешения конфликта при одновременном редактировании документа. Готовность к входу очередного участника в критическую секцию определяется по формуле 4.

$$E^{i}_{\text{готовКР}} = E^{i-1}_{\text{свобод}} \cup E^{i}_{\text{запрос}} \quad (4)$$

$E^{i}_{\text{готовКР}}$ – событие, определяющее состояние готовности входа очередного пользователя в критическую секцию;

$E^{i-1}_{\text{свобод}}$ – событие, проверяющее критический ресурс на занятость предыдущим пользователем;

$E^{i}_{\text{запрос}}$ – событие, определяющее, доступен ли критический ресурс очередному пользователю.

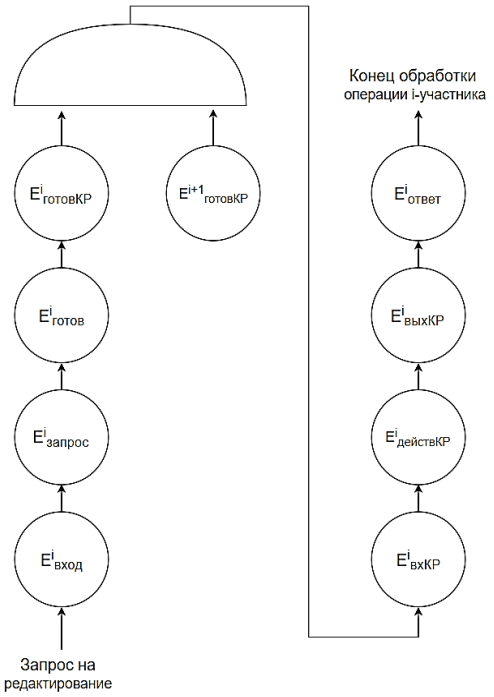

Рисунок 4 – Математическая модель механизма разрешения конфликтов

Параллельно с пользователем, редактирование локальной копии документа осуществляют удаленные участники. При каждом запросе удаленного участника на изменение документа, программа создает отдельный поток для редактирования. Если критическая секция свободна, то осуществляется операция над документом с последующей обработкой списка операций, стоящих в очереди на обработку.

Заключение. В ходе анализа существующих решений было выявлено, что самым часто используемым методом реализации является использование алгоритмов *CRDT* и метод блокирования сегмента документа. Как говорилось ранее, эти методы самые простые в реализации, позволяют по-своему сохранить целостность документа и позволяют довольно быстро создать качественный продукт. Использование древовидных структур с *zipper*'ами встречается намного реже из-за сложности передачи таких структур по вычислительной сети. Метод ОП считается устаревшим, большинство решений на ОП перешло к более простому и надежному методу с использованием *CRDT*.

Работа выполнена при поддержке гранта РФФИ «Конкурс на лучшие проекты фундаментальных научных исследований», грант № 19-07-00516 А.

СПИСОК ЛИТЕРАТУРЫ:

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ, Вильямс Издательский дом – 2013. – №3. – С. 140-173.

2. Вашкевич Н. П. Недетерминированные автоматы в проектировании систем параллельной обработки. – 2004. – С. 5.

3. Кнут Д. Искусство программирования т.1. Основные алгоритмы //Вильямс Издательский дом 2000. – С. 27.

4. Синев, М. П. Последовательный алгоритм обработки данных при анализе действий операторов // Новые информационные технологии и системы: сб. тр. 10-й Междунар. науч.-техн. конф. – Пенза, 2011. – С. 220 –223.

5. Макконнелл С. Совершенный код: Практическое руководство по разработке программного обеспечения //Русская редакция/БХВ - 2017. - С. 126 -129.

6. Бабинский А.З., Букатов А.А., Шапиро В.А. Определение базовых сервисов, разработка методики наполнения и методов реализации образовательных порталов //- 2003. - credit.edu.ru. – С. 7.

7. Синев М. П. Проблемы динамического отображения информации в системах объективного контроля радиотехнического комплекса / Пащенко Д. В., Трокоз Д. А., Синев М. П. // Проведение научных исследований в области обработки, хранения, передачи и защиты информации: сб. науч. тр. – Ульяновск: УлГТУ, 2009. – № 4. – С. 150–154.

8. Fraser Neil Differential Synchronization – Amphitheatre Parkway Mountain View, CA, 94043 [Электронный ресурс]. URL https://neil.fraser.name/writing/sync/ (дата обращения: 04.10.2020).

9. Терехов А.Н., Романовский К.Ю., Д.В. Кознов, П.С. Долгов, А.Н. Иванов Real: методология и CASE-средство для разработки систем реального времени и информационных систем. [Электронный ресурс]. URL http://www.math.spbu.ru/user/dkoznov/papers/Real_General.pdf (дата обращения: 02.10.2020).

10. Пащенко Д. В., Синев М. П. Методика построения систем объективного контроля авиационных радиолокационных комплексов //Известия высших учебных заведений. Поволжский регион. Технические науки. – 2009. – №. 4. – С. 49-50.

11. Вашкевич Н. П., Бикташев Р. А. Достоинство формального языка, основанного на концепции недетерминизма, при структурной реализации параллельных систем логического управления процессами и ресурсами //Известия высших учебных заведений. Поволжский регион. Технические науки. – 2011. – №. 1. – С. 4-8.

12. Вашкевич Н. П., Бикташев Р. А., Синев М. П. Формализация алгоритмов управления многопоточным доступом к разделяемым ресурсам на основе использования событийных недетерминированных автоматов //Интеллект. Инновации. Инвестиции. – 2014. – №. 1. – С. 128-133.

13. Синев М. П. Автоматная модель межпоточного обращения к разделяемому ресурсу в алгоритмах объективного контроля / М. П.Синев, М. Н. Синева // Молодежь. Наука. Инновации: сб. тр. 7-й Междунар. науч.-практ. конф. [Электронный ресурс]. – Пенза: РГУИТП, 2013. URL: http://rgu-penza.ru/mni/content/files/2013_Sinev,%20Sineva.pdf (дата обращения: 05.10.2020).

14. Pashchenko D. et al. Formal transformation inhibitory safe Petri nets into equivalent not inhibitory //Procedia Computer Science. – 2015. – Т. 49. – С. 99-103.

15. Pashchenko D. et al. The methodology of multicriterial assessment of Petri nets' apparatus //MATEC Web of Conferences. – EDP Sciences, 2016. – Т. 44. – С. 01009.

16. Pashchenko D. V. et al. Directly executable formal models of middleware for MANET and Cloud Networking and Computing //Journal of Physics: Conference Series. – IOP Publishing, 2016. – Т. 710. – №. 1. – С. 012024.

17. Синев М. П. Реализация стратегий объективного контроля в радиолокационных комплексах / Д. В. Пащенко, М. П. Синев // Проведение научных исследований в области обработки, хранения, передачи и защиты информации: сб. науч. тр. – Ульяновск : УлГТУ, 2009. – № 4. – С. 155–158.

18. Синев М.П. Поиск решения в экспертных системах объективного контроля // Новые информационные технологии и системы: сб. тр. 10-й Междунар. науч.-техн. конф. – Пенза, 2011. – С. 185–187.

19. Синев М. П. Система настройки диагностики радиолокационных комплексов/Пащенко Д. В., Трокоз Д. А., Синев М. П. // Новые информационные технологии и системы: сб. тр. 8-й Междунар. науч.-техн. конф. – Пенза, 2008. – С. 290–295.

Статья поступила в редакцию 04.11.2020
Статья принята к публикации 11.12.2020