Программные системы и вычислительные методы
*Правильная ссылка на статью:*
Алпатов А.Н., Юров И.И. Алгоритм и программная реализация совместного редактирования графических схем в режиме реального времени с использованием библиотеки Socket.IO // Программные системы и вычислительные методы. 2024. № 1. DOI: 10.7256/2454-0714.2024.1.70173 EDN: PQMMUM URL: https://nbpublish.com/library_read_article.php?id=70173

# Алгоритм и программная реализация совместного редактирования графических схем в режиме реального времени с использованием библиотеки Socket.IO

**Алпатов Алексей Николаевич**

ORCID: 0000-0001-8624-1662

кандидат технических наук

доцент, кафедра инструментального и прикладного программного обеспечения, Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА — Российский технологический университет»

119454, Россия, Московская область, г. Москва, проспект Вернадского, 78, каб. Г-225

alpatov@mirea.ru

**Юров Илья Игоревич**

ORCID: 0009-0003-7121-4321

магистр, кафедра инструментального и прикладного программного обеспечения, Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА — Российский технологический университет»

109383, Россия, Московская область, г. Москва, ул. Полбина, 35к2, 912

frit_027@mail.ru

Статья из рубрики "Языки программирования"

**DOI:**

10.7256/2454-0714.2024.1.70173

**EDN:**

PQMMUM

**Дата направления статьи в редакцию:**

20-03-2024

**Дата публикации:**

31-03-2024

**Аннотация:** В современном мире командная работа становится все более распространенной. Разные участники могут находиться в разных местах, но им все равно нужно работать вместе над одним проектом, в том числе и над графическими схемами. Важным аспектом такого подхода является возможность наблюдать изменения, вносимые другими участниками, в режиме реального времени. Это позволяет, прежде всего, снизить частоту конфликтов при одновременном редактировании одного и того же элемента схемы. Однако существующие решения для обмена данными при совместном редактировании графических схем в режиме реального времени сталкиваются с рядом проблем, такими как задержки при передаче данных. Предметом исследования в настоящей статье является разработка минимально жизнеспособного веб-приложения, позволяющего пользователям осуществлять совместное графическое редактирование полотна в режиме реального времени. Объектом исследования выступает модель процесса совместного редактирования в реальном времени с учетом разрешения возникающих конфликтов. Методология исследования основана на теоретическом подходе по выявлению математических формул, описывающих изменение состояния документа при его совместном редактировании пользователями. Дана характеристика применения протоколов HTTP и WebSocket в многопользовательских клиент-серверных приложениях. Для применения протокола WebSocket используется библиотека Socket.IO. Сервер приложения построен с помощью фреймворка Express. Основным вкладом авторов в исследование темы является модель процесса совместного редактирования в реальном времени, а также механизм определения конфликтов для любого количества пользователей и функция разрешения конфликтов для каждой пары конфликтующих изменений при совместном редактировании документов в режиме онлайн. В рамках данного исследования дополнительно предложен алгоритм совместного редактирования графических схем в режиме реального времени и дана его реализация в виде программной системы. Предложенный в результате исследования алгоритм на языке программирования JavaScript может быть использован в качестве основы для разработки более многофункциональных веб-приложений с использованием библиотеки Socket.IO и являться объектом будущих исследований, затрагивающих многопользовательское взаимодействие и разрешение конфликтов в режиме реального времени.

**Ключевые слова:**

протокол HTTP, протокол WebSocket, клиент-серверное приложение, совместное редактирование, определение конфликтов, разрешение конфликтов, язык программирования JavaScript, графическая схема, алгоритм, управляемость событиями

## Введение

Протокол HTTP появился в конце XX века и за это время стал преобладающей технологией передачи данных, применяемой при разработке клиент-серверных приложений в веб-среде [1]. В основе работы протокола HTTP лежит принцип «запрос-ответ», когда клиент посылает нужный запрос, например, GET или POST, а сервер в зависимости от типа запроса выполняет необходимые действия и возвращает клиенту нужные данные. Для реализации алгоритма совместного редактирования графических схем в режиме реального времени в протоколе HTTP выявляются существенные недостатки: отсутствие двунаправленной и полнодуплексной связи между компонентами

системы [2]. Вследствие этого процесс обмена координатами курсора мыши между сервером и клиентами при рисовании линий в браузере может вызвать задержки и оказывать дополнительную нагрузку на сервер.

Обозначенных недостатков лишен протокол WebSocket. Технология позволяет веб-серверу и браузеру обмениваться данными по одному и тому же соединению одновременно в обоих направлениях [3]. Сервер в любой момент времени может отправить сообщение сразу всем клиентам, ранее установившим с ним соединение. Точно также любой клиент по тому же соединению может отправить нужные данные на сервер [4].

Но при использовании протокола WebSocket без вспомогательных библиотек реализация алгоритма совместного редактирования графических схем в режиме реального времени потребует разработки дополнительных процедур для обеспечения бесперебойной и стабильной работы компонентов системы. В рамках данной работы будет предложено решение данной задачи за счет использования JavaScript-библиотеки Socket.IO, которая добавляет управляемость событиями, автоматическое переподключение при разрыве соединения, мультиплексирование и другие функции [5].

**Модель процесса совместного редактирования в реальном времени**

Пусть есть два пользователя, A и B, которые работают над одной графической схемой в разных процессах. Пусть D(t) представляет текущее состояние документа в момент времени t в процессе совместного редактирования графической схемы. Предположим, что каждый пользователь вносит изменения в документ независимо от других пользователей, и все изменения синхронизируются в реальном времени.

Тогда процесс совместного редактирования можно описать следующей формулой:

$$D(t) = D_0 + \Delta D_1(t) + \Delta D_2(t) + \cdots + \Delta D_n(t), \quad (1)$$

где $D_0$ — начальное состояние документа до начала редактирования, $\Delta D_i(t)$ — изменения, внесенные i-ым пользователем в момент времени t, n — количество пользователей, участвующих в редактировании.

Таким образом, текущее состояние документа D(t) представляет собой сумму начального состояния документа и всех изменений, внесенных каждым пользователем в реальном времени. Такой подход отражает динамику процесса совместного редактирования графической схемы в веб-приложениях и учитывает вклад каждого участника в формирование итогового состояния документа.

В ходе совместного редактирования документов в режиме онлайн могут возникать различные конфликты (конфликт вставки, удаления, форматирования, перемещения и т. д.) [6]. Рассмотрим процесс возникновения конфликтов данных в веб-приложении для совместного редактирования документов. Пусть каждый пользователь, редактирующий графическую схему, имеет свой локальный набор данных о состоянии документа. Пусть $D_A(t)$ и $D_B(t)$ представляют текущее состояние документа для пользователя A и пользователя B в момент времени t. Тогда можно определить конфликт следующим образом: конфликт возникает, если пользователя A и B внесли изменения в один и тот же узел или связь графической схемы в период времени Δt — интервал времени, в течение которого изменения могут считаться конфликтными.

Иными словами, оба пользователя начинают редактировать документ в момент времени $t_0$. Пользователь A внес изменения в документ, приводя к изменению его состояния от $D(t_0)$ до $D_A(t_1)$ в момент времени $t_1$, или $D_A(t_1) = D(t_0) + \Delta D_A(t_1)$. Пользователь B также внес изменения в документ, что приводит к изменению его состояния от $D(t_0)$ до $D_B(t_2)$ в момент времени $t_2$ или $D_B(t_1) = D(t_0) + \Delta D_B(t_2)$. Теперь если $t_1 < t_2$, то возникает конфликт, так как оба пользователя внесли изменения в документ в разное время.

Формула определения конфликтов времени редактирования может быть записана следующим образом:

$$\Psi(D_A(t), D_B(t), \Delta_t) = \{(n, t) \mid n, t \in [t, t + \Delta t]\}, \quad (2)$$

где n — узел или связь, (n, t) представляет изменение, внесенное пользователем в узел или связь n в момент времени t.

Формула определения конфликтов для любого количества пользователей n может быть записана следующим образом:

$$\Psi\left(D_{U_1}(t), D_{U_2}(t), \ldots D_{U_n}(t), \Delta_t\right) = \{(n, t) \mid n, t \in [t, t + \Delta t]\}, \quad (3)$$

где n — узел или связь.

Если для пользователей A и B обнаружены одинаковые изменения в одном и том же узле или связи в течение интервала Δt, то это считается конфликтом редактирования.

Пусть

$$D_{U_i}(t) = \left\{\delta_{i,1}(t), \delta_{i,2}(t), \ldots, \delta_{i,k_i}(t)\right\} \quad (4)$$

— это набор изменений, внесенныхпользователем $U_i$ в момент времени t, где $k_i$ — количество изменений.

Для каждого изменения $\delta_{i,j}(t)$ определим его совместимость с текущим состоянием документа как $C_{i,j} = f(\delta_{i,j}(t), D(t))$. Тогда конфликт совместимости изменений (два изменения несовместимы) определяется через

$$\Psi_{i,j,k,l}(t) \begin{cases} 1, \text{если } C_{i,j}(t) \neq C_{k,l}(t) \\ 0, \text{иначе} \end{cases}. \quad (5)$$

Тогда для каждой пары конфликтующих изменений можно применить функцию разрешения конфликтов вида

$$\Theta_{i,j,k,l}(t) = g\left(\delta_{i,j}(t), \delta_{k,l}(t), D(t)\right), \quad (6)$$

где $\Theta_{i,j,k,l}(t)$ — результат разрешения конфликта между изменениями $\delta_{i,j}(t)$ и $\delta_{k,l}(t)$ в момент времени t, $\delta_{i,j}(t)$ и $\delta_{k,l}(t)$ — изменения, которые конфликтуют между собой, D(t) представляет текущее состояние документа в момент времени t.

Функция g применяется для разрешения конфликта между этими двумя изменениями. Она принимает два конфликтующих изменения $\delta_{i,j}(t)$ и $\delta_{k,l}(t)$ а также текущее состояние документа D(t) и возвращает разрешенное изменение. Ее задача — определить, какое из этих изменений следует применить к документу. Функция g может быть реализована с

использованием различных алгоритмов и стратегий разрешения конфликтов и в зависимости от характера конфликта. Например, если изменения $\delta_{i,j}(t)$ и $\delta_{k,l}(t)$ не взаимоисключающие, функция g может объединить их в одно изменение. Практически это может быть использовано для разрешения ситуации, когда одно изменение вставляет текст, а другое изменение добавляет форматирование этого текста, их можно объединить в одно изменение, которое вставляет текст с форматированием. Так $\delta_1(t)$ представляет изменение, вставляющее текст $Text_1$ в позицию $Pos_1$, а $\delta_2(t)$ — изменение, добавляющее форматирование $Format_2$ к этому тексту. Тогда функция g, примет следующий вид:

$$g\left(\delta_{i,j}(t), \delta_{k,l}(t), D(t)\right) = \varphi_{\text{merged}}(t), \quad (7)$$

где $\varphi_{merged}(t)$ — объединенное изменение, которое вставляет текст $Text_1$ с применением форматирования $Format_2$ в позицию $Pos_1$.

**Алгоритм совместного редактирования графических схем в режиме реального времени**

Реализацию алгоритма следует начать с написания серверной части приложения. Сервер с нужным портом можно создать различными способами, но в данном исследовании использовался минималистичный фреймворк Express [7]. После этого следует инициализировать экземпляр io с помощью класса Server, предоставляемый библиотекой Socket.IO, передав в его конструктор ранее созданный объект сервера [8]. Получив WebSocket-сервер, необходимо добавить слушателей событий и обработчики для них. В данном алгоритме сервер ожидает два события: connection и draw. При возникновении первого события сервер генерирует событие color и передает вновь подключившемуся клиенту случайный HEX-код цвета для графических линий пользователя. При возникновении события draw сервер отправляет полученные координаты точки перемещения курсора мыши конкретного пользователями всем остальным клиентам системы по тому же соединению. На рисунке 1 представлен код описанной части алгоритма с экземпляром io.

```
37  const io : Server<DefaultEventsMap, DefaultEventsMap, DefaultEventsMap, any> = new Server(server);
38
39  io.on( ev: 'connection', listener: (socket : Socket<DefaultEventsMap, DefaultEventsMap, DefaultEventsMap, any> ) : void => {
40      socket.emit( ev: 'color', args: { color: getColor() });
41
42      socket.on( ev: 'draw', listener: (data) : void => {
43          socket.broadcast.emit( ev: 'draw', data);
44      });
45  });
```

Рисунок 1. Серверная часть алгоритма с экземпляром io

В клиентской части алгоритма в первую очередь следует инициализировать экземпляр socket и подписать его на два события: color и draw. В обработчике первого события в переменную strokeStyle будет сохраняться цвет, полученный с сервера. В свою очередь, в обработчике draw клиент будет получать с сервера координаты курсора мыши другого пользователя и строить по ним графическую линию на своем экране с помощью специальной функции drawLine(). Стоит отметить, что сама линия отображается с помощью стандартных средств языка JavaScript, а именно, инструмента Canvas API [9].

Затем нужно проинициализировать необходимые переменные начальными значениями, получить элемент canvas из HTML-кода страницы и привязать к нему три события:

mousedown (пользователь нажал мышью по графической области), mousemove (пользователь двигает мышью с зажатой кнопкой) и mouseup (пользователь опустил кнопку мыши) [10]. У каждого из указанных событий есть свой обработчик. При возникновении события mousedown алгоритм сохраняет начальные координаты startX и startY курсора мыши в созданный ранее объект mouse и устанавливает переменную isDraw в значение true. При событии mousemove, если значение isDraw истинно, сохраняются следующие координаты endX и endY курсора мыши, после чего все данные отправляются на сервер по соединению веб-сокета события draw и строится графическая линия текущего пользователя.

Затем в свойства объекта mouse startX и startY записываются значения endX и endY соответственно. На рисунке 2 продемонстрирован код обработчика события mousemove.

```
canvas.addEventListener( type: 'mousemove', listener: (e : MouseEvent ) : void => {
    if (isDraw) {
        const rect : DOMRect = e.target.getBoundingClientRect();
        mouse.endX = e.pageX - rect.left;
        mouse.endY = e.pageY - rect.top;

        const coordinates : {…} = {
            startX: mouse.startX,
            startY: mouse.startY,
            endX: mouse.endX,
            endY: mouse.endY,
        };

        const data : {…} = { ...coordinates, strokeStyle }
        socket.emit( ev: 'draw', data);
        drawLine(context, data);

        mouse.startX = mouse.endX;
        mouse.startY = mouse.endY;
    }
});
```

Рисунок 2. Код обработчика события mousemove

При возникновении последнего события mouseup переменная isDraw устанавливается в значение false.

После локального запуска сервера можно проверить работу алгоритма, нарисовав несколько простых геометрических фигур в трех разных браузерах. Результат показан на рисунке 3.

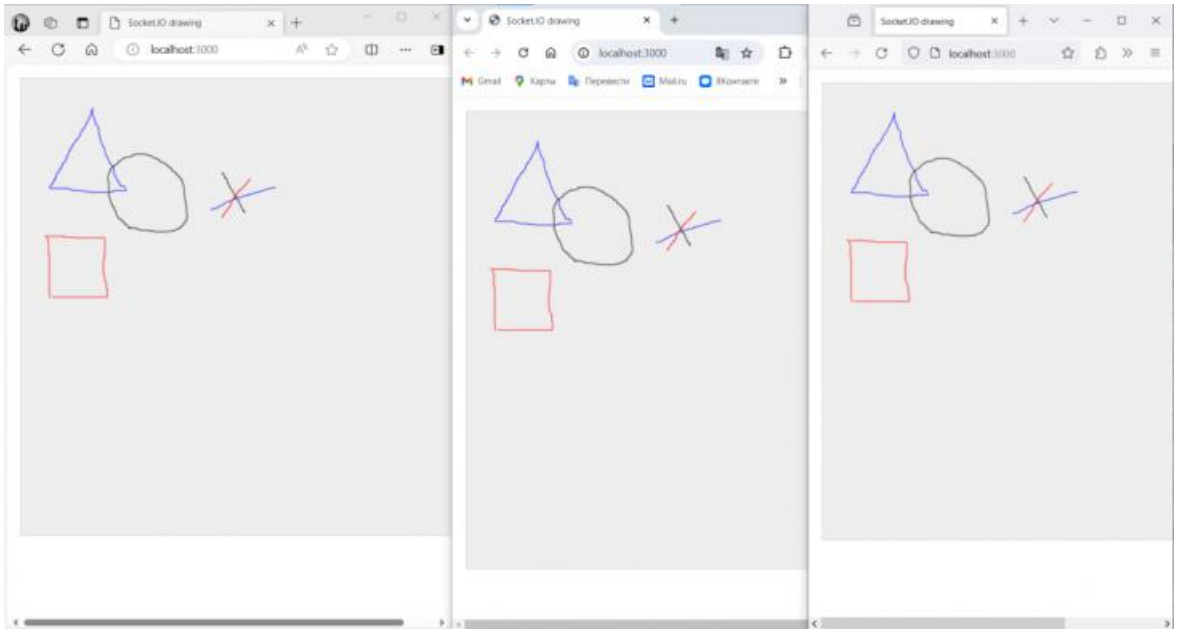

Рисунок 3. Результат работы алгоритма

**Заключение**

Можно сделать вывод, что с помощью библиотеки Socket.IO возможна реализация приложений совместного редактирования графических схем в режиме реального времени на основе описанного алгоритма. Предложенный в данной работе метод показал свою стабильность работы и удобство реализации. Стоит отметить, что данный алгоритм ожидает доработок в том случае, если, например, необходимо рисовать ровные графические фигуры одним движением мыши, что потребует дополнительных будущих исследований. Тем не менее, полученный алгоритм является удобной основой для более сложных программ.

## Библиография

1. Князев А. А., Кондратьев А. Н., Дубровский Н. С. Эволюция и особенности протокола HTTP // Инновационный потенциал развития общества: взгляд молодых ученых: сборник научных статей 4-й Всероссийской научной конференции перспективных разработок, Курск, 01 декабря 2023 года. – Курск: ЗАО «Университетская книга», 2023. – С. 176-178.
2. Kovaliuk, D., Kovaliuk, O.O., Pinaieva, O., Kotyra, A., & Kalizhanova, A. (2019). Optimization of web-application performance. *Symposium on Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments (WILGA)*.
3. Gursesli, M.C.; Selek, M.E.; Samur, M.O.; Duradoni, M.; Park, K.; Guazzini, A.; Lanatà, A. Design of Cloud-Based Real-Time Eye-Tracking Monitoring and Storage System. Algorithms 2023, 16, 355. https://doi.org/10.3390/a16070355
4. Горчаков А. Я. Разработка клиентской архитектуры системы мгновенных сообщений по технологии WebSocket // Гагаринские чтения-2018: Сборник тезисов докладов XLIV Международной молодёжной научной конференции, Москва-Байконур-Ахтубинск, 17–20 апреля 2018 года. Том 2. – Москва-Байконур-Ахтубинск: Московский авиационный институт (национальный исследовательский университет), 2018. – С. 209.
5. Шабанов А. Э. Обзор библиотеки socket.io // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. – 2022. – № 1(35). – С. 56-62.

6. Царева Е. В. Разрешение конфликтных ситуаций при синхронизации многопользовательских online-приложений // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. – 2016. – № 1(34). – С. 79-91.
7. Чернышев Я. Р. Разработка и самостоятельный хостинг веб-приложения на основе фреймворка Express.js // Студенческая наука-взгляд в будущее: Материалы XVIII Всероссийской студенческой научной конференции, Красноярск, 15–17 марта 2023 года. Том Часть 5. – Красноярск: Красноярский государственный аграрный университет, 2023. – С. 291-293.
8. Karam, Sameer & Abdulrahman, Bikhtiyar. (2022). Using Socket.io Approach for Many-to-Many Bi-Directional Video Conferencing. AL-Rafidain Journal of Computer Sciences and Mathematics. 16. 81-86. 10.33899/csmj.2022.174411.
9. Macklon, Finlay & Viggiato, Markos & Romanova, Natalia & Buzon, Chris & Paas, Dale & Bezemer, Cor-Paul. (2023). A Taxonomy of Testable HTML5 Canvas Issues. IEEE Transactions on Software Engineering. PP. 1-13. 10.1109/TSE.2023.3270740.
10. Кочитов М. Е. Рисование компьютерной мышью на холсте веб-страницы браузера с помощью инструмента Canvas // Постулат. – 2019. – № 8(46). – С. 25

## Результаты процедуры рецензирования статьи

*В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.*
*Со списком рецензентов издательства можно ознакомиться здесь.*

В статье авторы подробно излагают проблематику существующих методов взаимодействия в веб-приложениях, акцентируя внимание на недостатках протокола HTTP для задач совместной работы и предлагая в качестве решения использование WebSocket с библиотекой Socket.IO. Методология исследования представлена разработкой модели процесса совместного редактирования и алгоритма, реализующего эту модель в реальном времени. Особое внимание уделено обработке конфликтов при одновременном внесении изменений разными пользователями. Методология объединяет теоретический анализ и практическую разработку, демонстрируя на примере разработки веб-приложения. Актуальность исследования очевидна, учитывая растущую потребность в инструментах для совместной работы в режиме онлайн. Современные веб-технологии предлагают новые возможности для разработки подобных приложений, и поиск оптимальных решений в этой области представляет большой интерес. Научная новизна заключается в разработке конкретного алгоритма с использованием библиотеки Socket.IO для решения задачи совместного редактирования графических схем. Особенностью является подробное описание механизма обработки конфликтов, возникающих при одновременном редактировании одного документа. Статья имеет четкую структуру, включая введение, описание проблемы, разработку модели процесса, описание алгоритма и заключение. Стиль изложения научный, содержание подается логично и последовательно. Иллюстрации и схемы эффективно дополняют текст, способствуя лучшему пониманию материала. Библиография содержит актуальные источники, что свидетельствует о тщательном подходе авторов к изучению предмета. Список литературы представляет собой смесь работ по теоретическим основам и практическим разработкам в области веб-технологий. Статья будет интересна как исследователям в области компьютерных наук, так и практикующим разработчикам, заинтересованным в создании совместных редакторов и приложений реального времени. Выводы подчеркивают потенциал использования библиотеки Socket.IO для

решения задачи совместного редактирования графических схем, а также указывают на возможности дальнейших исследований в этом направлении, включая оптимизацию производительности и улучшение пользовательского интерфейса. Авторы уверенно демонстрируют, как их разработка может служить основой для более сложных и функциональных систем совместной работы. Также они отмечают необходимость дальнейших исследований для решения специфических задач, например, реализации более сложных графических фигур и интеграции с другими инструментами взаимодействия в реальном времени. Статья представляет значительный интерес для научного сообщества и практикующих специалистов в области разработки программного обеспечения для совместной работы. Материал изложен доступно, аргументированно и снабжен необходимыми примерами и иллюстрациями. Однако, авторам могут быть предложены к доработке следующие аспекты: более подробное обсуждение потенциальных ограничений предложенного решения и более глубокий анализ существующих альтернативных подходов к совместному редактированию в режиме реального времени. Это позволит усилить позицию работы в контексте современных исследований и разработок. Также рекомендую разбить рисунок 4 на несколько частей (в виде отдельных рисунков) с их детальным описанием. Исходя из вышеизложенного, рекомендую статью к публикации после выполнения незначительных доработок, указанных выше, для дальнейшего усиления аргументации и обоснованности выбранных решений.

## Результаты процедуры повторного рецензирования статьи

*В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.*
*Со списком рецензентов издательства можно ознакомиться здесь.*

Предметом исследования в рецензируемой статье выступает алгоритм и программная реализация совместного редактирования графических схем в режиме реального времени с использованием библиотеки Socket.IO.
Методология исследования базируется на обобщении сведений из научных публикаций по изучаемой теме, программной реализации предлагаемого алгоритма.
Актуальность работы авторы связывают с наличием существенных недостатков в протоколе HTTP для реализации алгоритма совместного редактирования графических схем в режиме реального времени (отсутствие двунаправленной и полнодуплексной связи между компонентами системы), из-за которых процесс обмена координатами курсора мыши между сервером и клиентами при рисовании линий в браузере может вызвать задержки и оказывать дополнительную нагрузку на сервер.
Научная новизна рецензируемого исследования, по мнению рецензента, состоит в предложениях по использованию JavaScript-библиотеки Socket.IO, которая при совместном редактировании графических схем добавляет управляемость событиями, автоматическое переподключение при разрыве соединения, мультиплексирование и другие функции.
В статье структурно выделены следующие разделы: Введение, Модель процесса совместного редактирования в реальном времени, Алгоритм совместного редактирования графических схем в режиме реального времени, Заключение, Библиография.
Авторы рассматривают ситуацию, когда два пользователя работают над одной графической схемой в разных процессах, каждый пользователь вносит изменения в документ независимо от других пользователей, и все изменения синхронизируются в реальном времени. В публикации приведена серверная часть алгоритма, реализованная

с использованием минималистичного фреймворка Express. По мнению авторов, применение протокола WebSocket позволяет веб-серверу и браузеру обмениваться данными по одному и тому же соединению одновременно в обоих направлениях, сервер в любой момент времени может отправить сообщение сразу всем клиентам, ранее установившим с ним соединение. В результате исследования авторы приходят к выводу о том, что с помощью библиотеки Socket.IO возможна реализация приложений совместного редактирования графических схем в режиме реального времени на основе описанного алгоритма.

Библиографический список включает 10 источников – публикации отечественных и зарубежных ученых по теме статьи, на которые в тексте имеются адресные ссылки, подтверждающие наличие апелляции к оппонентам.

Из существенных недостатков публикации следует отметить, что рисунок 4 не читается, а судя по тексту статьи именно на нем отражены итоги исследования в виде обобщенной схемы работы предложенного алгоритма совместного редактирования графических схем в режиме реального времени.

Статья отражает результаты проведенного авторами исследования, соответствует направлению журнала «Программные системы и вычислительные методы», содержит элементы научной новизны и практической значимости, может вызвать интерес у читателей, но нуждается в доработке в соответствии с высказанным замечанием.