ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2016 Управление, вычислительная техника и информатика № 1 (34)

УДК 004.021:004.738.2
DOI: 10.17223/19988605/34/9

Е.В. Царева

РАЗРЕШЕНИЕ КОНФЛИКТНЫХ СИТУАЦИЙ ПРИ СИНХРОНИЗАЦИИ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ONLINE-ПРИЛОЖЕНИЙ

Приведено сравнение конфликтных ситуаций для задачи синхронизации многопользовательских online-приложений и других задач синхронизации. Предложены алгоритмы разрешения конфликтов на стороне клиента. Предлагается методика синхронизации приложений на основе указанных алгоритмов, в том числе способ программной реализации. Рассмотрена синхронизация flash-приложения «Мозговой штурм» в системе вебинаров Adobe Connect Meeting 9.

Ключевые слова: синхронизация; многопользовательское приложение; конфликт доступа; блокировка; совместное редактирование.

Термин «синхронизация» сегодня применяется чрезвычайно широко и означает приведение некоторых процессов к одному виду, совместное использование ресурсов, обмен сообщениями между процессами. Термин применяется при обеспечении многозадачности в операционных системах, в многопоточном и параллельном (на многих процессорах) выполнении задач, в базах данных для приведения двух копий базы данных к одной версии или при выполнении конфликтующих транзакций, при приведении зеркальных копий какого-либо сервера к одной версии.

При организации сетей ЭВМ термин применяется на уровне передачи данных (сверка правильности доставленных данных), при синхронизации времени на узлах в сети ЭВМ. Синхронизацией называется приведение к одному виду большого объема данных, например в мобильных сетях, когда при подключении мобильной платформы в сеть происходит синхронизация измененных данных с удаленным сервером (отправка изображений на сервис хранения личной информации, обновление комментариев к новости и др.) [1]. Синхронизацией называется передача больших объемов данных по технологии BitTorrent [2].

Термин сетевой синхронизации применяется также в случае работы многопользовательского online-приложения, когда внесенные несколькими пользователями изменения в рабочую область сразу же (в режиме реального времени) отображаются на экране других пользователей. Примером могут служить совместное редактирование диаграмм и документов, проведение виртуальных совещаний (вебинаров), многопользовательские игры, виртуальные миры.

Перечисленные примеры могут быть отнесены к синхронизации данных (приведение к одному виду двух наборов данных) или к синхронизации процессов (организация непротиворечивого выполнения нескольких процессов). Синхронизируемые процессы могут происходить в системе с разделяемой или распределенной памятью, а синхронизация данных может выполняться однократно по запуску [3] или в режиме реального времени. Далее сравним особенности синхронизации многопользовательских приложений с перечисленными подвидами задачи синхронизации относительно основной проблемы синхронизации – разрешения конфликтных ситуаций.

Если при синхронизации процессов используется общая память, то разработчик должен обеспечить бесконфликтное использование общих переменных, блокируя их на время использования, или, если выражаться точнее, блокируя остальные процессы-претенденты на время выполнения операции. В базах данных для этого предусмотрены блокировки на чтение и запись, а при многопоточном программировании используются семафоры и основанные на них мьютексы, рандеву, барьеры и другие конструкции [4]. Так как процессы могут друг друга блокировать, то возможна ситуация взаимной блокировки, например, когда два процесса обращаются к двум ресурсам в разном порядке.

Для синхронизации многопользовательских приложений проблема блокировок не стоит так остро. Если клиентские и серверные приложения однопоточные, то одновременная работа с общей памятью для них невозможна и, значит, невозможны конфликты записи или чтения. В случае многопоточной архитектуры приложений проблемы общего использования памяти решаются на стороне клиента или сервера и не относятся к сетевой синхронизации приложения в целом.

Конечно, пользователи могут вносить изменения в приложение одновременно, создавая конфликт, однако заявка на блокировку объекта создаст дополнительную временную задержку, а блокировать приложение пользователя на время выполнения операции и вовсе недопустимо. Поэтому при возникновении конфликта все процессы, кроме выбранного, не блокируются, а отменяются. Правило выбора при этом становится концептуально важным и может зависеть только от предметной области синхронизируемого приложения и различаться для разных объектов.

В связи с этим возникают две особенности синхронизации приложения. Во-первых, это дополнительный тип конфликта, когда для двух связанных событий предусмотрены разные правила выбора и при их появлении выбираются одновременно два пользователя. Например, если движению объекта неразрывно предшествует его выделение, но при одновременном выделении объекта выбирается последний пользователь, а при одновременном движении объекта выбирается первый пользователь. Во-вторых, применить правило выбора пользователя в полной мере может только сервер, решение которого, впрочем, может быть неверным, так как какое-либо сообщение серверу еще не поступило. Поэтому клиентские приложения (имеются в виду авторы событий) должны уметь отменять свои решения (которые они выполнили для удобства пользователя, не дожидаясь ответа сервера) или отвергать поступающие сообщения до получения соответствующей инструкции сервера.

Пользователь может заблокировать один из объектов приложения, например для его перемещения, однако ни отсутствие этой блокировки, ни взаимная блокировка не становятся критически важными, поскольку пользователи в таких достаточно редких ситуациях смогут идентифицировать проблему и договориться об использовании объектов. Программное решение этих проблем является только вопросом комфорта (что, разумеется, тоже важно), но не вопросом безопасности используемой информации, как, например, в базах данных.

При синхронизации процессов с распределенной памятью (многопроцессорные кластеры) процессы обмениваются сообщениями, используя функции «отправить» и «получить» [5]. На время выполнения этих функций блокируется либо процесс, либо только буфер передачи данных (так называемый асинхронный вызов, при котором возможно продолжение работы программы). Взаимная блокировка возможна, если два процесса бесконечно ожидают получения или отправки данных.

Подобные функции также используются при передаче данных по сети на транспортном уровне (используются, например, протоколы TCP, UDP) [6]. Для исключения взаимной блокировки для чтения и отправки данных в системе выделяются отдельные потоки, которые при наступлении событий вызывают соответствующие функции-обработчики (слушатели) [7].

Сходство синхронизации процессов с распределенной памятью и синхронизации многопользовательского приложения состоит в использовании сообщений. Действительно, только посредством сообщений серверное приложение может обмениваться данными с клиентами. Различие такое же, как и было описано выше, состоит в разном понимании конфликтных ситуаций и использования блокировок.

Синхронизируемое многопользовательское приложение является приложением прикладного уровня, поэтому конфликты, возможные на транспортном уровне, решаются за счет протоколов более низкого уровня. Отметим, что протокол TCP обеспечивает единый порядок поступления сообщений, так что при его использовании все клиенты получат сообщения сервера если и не в одно и то же время, то как минимум в одном порядке, что сможет заменить временные метки, которые потребуются при использовании менее надежных протоколов.

Рассмотрим подвид задачи синхронизации – синхронизацию данных. Различие синхронизации данных с однократным выполнением и синхронизации многопользовательских приложений очевидно: в первом случае выполняется один процесс над двумя копиями одной модели предметной области по заранее заданным правилам, во втором случае происходит множество процессов, которые к тому же не детерминированы, так как выполняются пользователями. Сходство заключается в потенциальном мно-

гообразии правил, применяемых при возникновении конфликтных ситуаций, и в подходе, что только один вариант какого-либо объекта должен быть в итоге принят как верный.

Синхронизацию данных в режиме реального времени кратко рассмотрим на примере распределенных баз данных [8]. Распределенные базы данных состоят из нескольких узлов, каждый из которых обслуживает некоторое количество пользователей и которые периодически обмениваются обновленной информацией. Обмен может производиться журнальным методом, при котором на каждом узле в специальном журнале сохраняются все внесенные изменения, содержимое журнала периодически отправляется другим узлам системы, где при возникновении конфликтов применяются единые для всей системы правила [9]. Также в узлах системы может использоваться кэш, в котором хранится ограниченный набор используемых данных. Кэш может обновляться с каждой транзакцией (активный кэш) либо ждать появления несоответствующих данных для выполнения запроса у других узлов системы (пассивный кэш) [10].

Несмотря на сходство задачи синхронизации распределенных баз данных с задачей синхронизации многопользовательских приложений, различия между ними все же имеются. Многопользовательские приложения не обязательно подразумевают распределенное хранение данных, сервер в системе может быть один. Данные системы в конечном итоге хранятся не в виде записей, а визуализируются в клиентском приложении, функциональные зависимости между ними обеспечиваются конечным приложением и не создают дополнительных проблем, как при синхронизации распределенных баз данных. В силу трудоемкости разработки каждого события объем хранимой информации в многопользовательских приложениях будет существенно меньше, чем в среднестатистических базах данных.

Распределенные базы данных, в которых есть сильные функциональные связи, чтобы не блокировать на время редактирования большое число объектов, выполняют транзакции на некотором срезе данных; при возникновении конфликта в таких базах данных транзакции откатываются целиком. И хотя откат транзакции отдаленно схож с необходимостью отменять свои действия в клиентских приложениях, он предполагает повтор этой операции, возможно, в автоматическом режиме, что увеличивает временные задержки. Кроме того, распределенные базы данных не требуют ввода каких-либо условий, согласно которым узлы системы отвергают сообщения сервера до определенного момента. То есть научные публикации по вопросам синхронизации распределенных баз данных освещают несколько иные проблемы, чем те, которые возникают при синхронизации многопользовательских приложений.

Таким образом, синхронизация многопользовательских приложений стоит обособленно в ряду других подобных задач. Для определения подвида рассматриваемой задачи примем во внимание, что многопоточное программирование обычно используется для вычисления наукоемких задач или при организации автоматизированных систем управления, т.е. в случаях, когда заранее детерминированные процессы должны работать согласованно. Поскольку поведение пользователей заранее непредсказуемо, а задача состоит в поддержании одного состояния предметной области на всех клиентских приложениях, то сделаем вывод, что синхронизация многопользовательских онлайн-приложений является синхронизацией данных в режиме реального времени.

Синхронизируемые многопользовательские приложения можно подразделить на задачи совместного редактирования и онлайн-игры. Различия заключаются в том, что при совместном редактировании менее важны временные задержки, нет соперничества (т.е. необходимости справедливо отдать предпочтение одному из игроков в конфликтных ситуациях), пользователи редактируют одни и те же объекты, в то время как в играх обрабатываются их взаимодействия (столкновения, выстрелы и др.). Конфликтной ситуацией в онлайн-играх становится не одновременное использование ресурсов, а разное (в силу временных задержек) развитие сюжета в разных клиентских приложениях. Для разрешения подобных конфликтных ситуаций используется предсказывание движения игроков (например, на основе кубических сплайнов [11]), обман (когда приложение корректирует действия других игроков так, чтобы проигрыш пользователя выглядел реалистично), использование балластных действий на время ожидания отклика из сети, а также общее замедление игры [12]. Правила выбора наиболее правильного варианта могут быть такими же, как и при совместном использовании ресурсов: прав первый, прав последний, прав владелец объекта и, возможно, прав тот, в чьем приложении сложнее подделать реалистичную картину проигрыша.

В рассмотренной литературе концепция синхронизации многопользовательских приложений наиболее подробно рассматривается в публикациях, посвященных онлайн-играм, однако и в них не приводятся конкретные алгоритмические решения. Целью настоящей работы является предложение алгоритмов синхронизации событий многопользовательского приложения, предложение конкретных шагов по синхронизации произвольного онлайн-приложения, т.е. предложение методики синхронизации многопользовательских онлайн-приложений (в том числе формулируется алгоритм обработки поступающих событий в приложении клиента), а также применение методики на примере синхронизации flash-приложения «Мозговой штурм» в системе вебинаров Adobe Connect Meeting 9.

1. Алгоритмы разрешения конфликтов

Далее будем предполагать, что синхронизируемая система имеет клиент-серверную архитектуру, в которой клиент отправляет сообщение о произошедшем изменении серверу, а сервер рассылает полученную информацию другим клиентам, в том числе автору события, в неизменном виде (т.е. возвращает эхо). В качестве протокола связи используются надежные протоколы (TCP), которые обеспечивают порядок доставки сообщений. Обработка событий (в том числе разрешение конфликтов) происходит на стороне клиента, а сервер осуществляет хранение информации о текущем состоянии приложения, которое отправляет каждому вновь подключившемуся пользователю. Для разрешения конфликтов будут использоваться два способа: определение очередности поступления событий (важно первое или последнее событие) и блокировка используемого объекта.

Если важно первое событие, то все последующие аналогичные действия над объектами будут отклоняться (пока событие не будет отменено, например, выделение не будет сброшено). Если важно последнее, то, напротив, они будут приниматься к исполнению. Выполнение этого алгоритма очевидно для всех сторон работы многопользовательского приложения, кроме авторов событий, которые должны ориентироваться на ответ сервера (эхо своего запроса), чтобы определить, было ли их событие первым или последним в случае возникновения конфликта. Возвращаемое эхо собственного события при этом не несет какой-либо полезной нагрузки для приложения, вся информация о произошедшем событии хранится в памяти клиента, эхо используется только для определения очередности происходящих событий.

Схема обработки сообщений в приложении приведена на рис. 1.

Указанные алгоритмы позволяют отменить собственное синхронизируемое действие, если для него важно первое событие и ранее в сети произошло аналогичное событие, и отвергнуть сообщения о событиях, если для них важно последнее событие и они произошли раньше собственного.

Вторым способом предотвращения конфликтов является блокировка ресурсов на время редактирования одним из пользователей. События блокировки, изменения (редактирования) объекта и разблокировки, а также сброс выделения доступны только тому пользователю, который ранее объект выбрал (с помощью щелчка мыши или иным способом). Для этого при выделении объекта сохраняется идентификатор пользователя, который затем сравнивается с идентификаторами авторов поступающих событий. Такой порядок позволит избежать конфликт разных правил выбора для связанных событий, когда одно из них является, по сути, выделением, а второе – редактированием.

Если для события выделения объекта важно первое событие, то события блокировки и разблокировки можно не реализовывать, так как функции блокировки будут выполняться событиями выделения и сброса выделения. Если для события выделения важно последнее событие, то, поскольку выделить объект беспрепятственно может другой пользователь, события блокировки и разблокировки объекта на время редактирования объекта (например, движения) обязательны для реализации.

События редактирования объекта (во время его блокировки) и сброса выделения не требуют возвращения эха, так как идентификатор автора события однозначно определяет, допустимо ли событие к исполнению. События блокировки и разблокировки объекта должны происходить синхронно в приложении, поэтому они требуют возвращения эха.

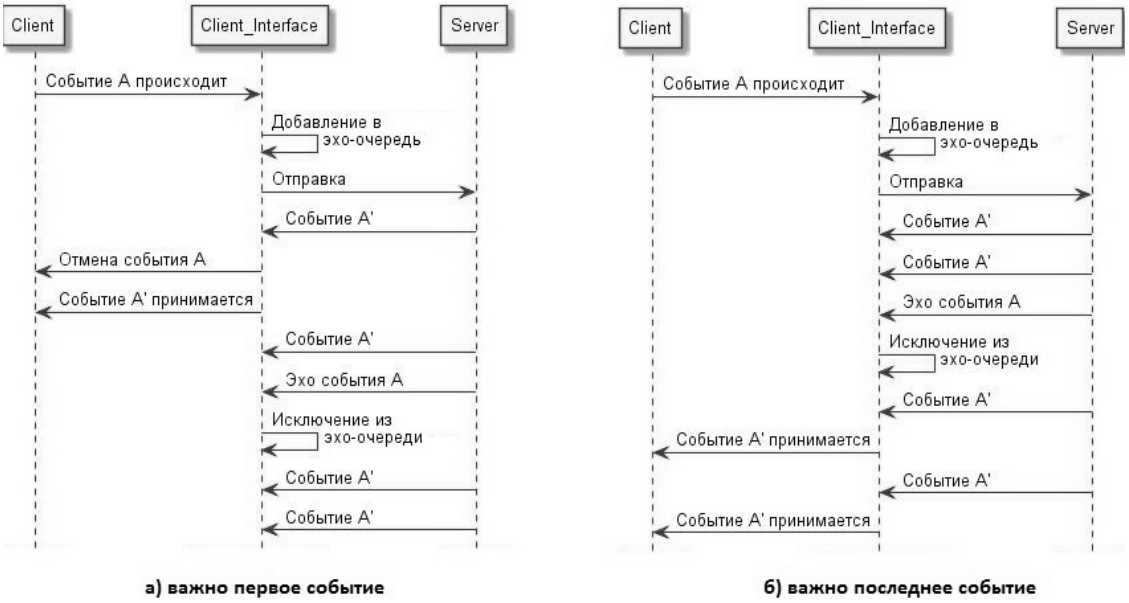

Рис. 1. Работа с событиями одного вида в приложении с учетом их очередности: *а* – важно первое событие; *б* – важно последнее событие. *A* – собственное событие клиента; *A'* – аналогичное сообщение другого пользователя

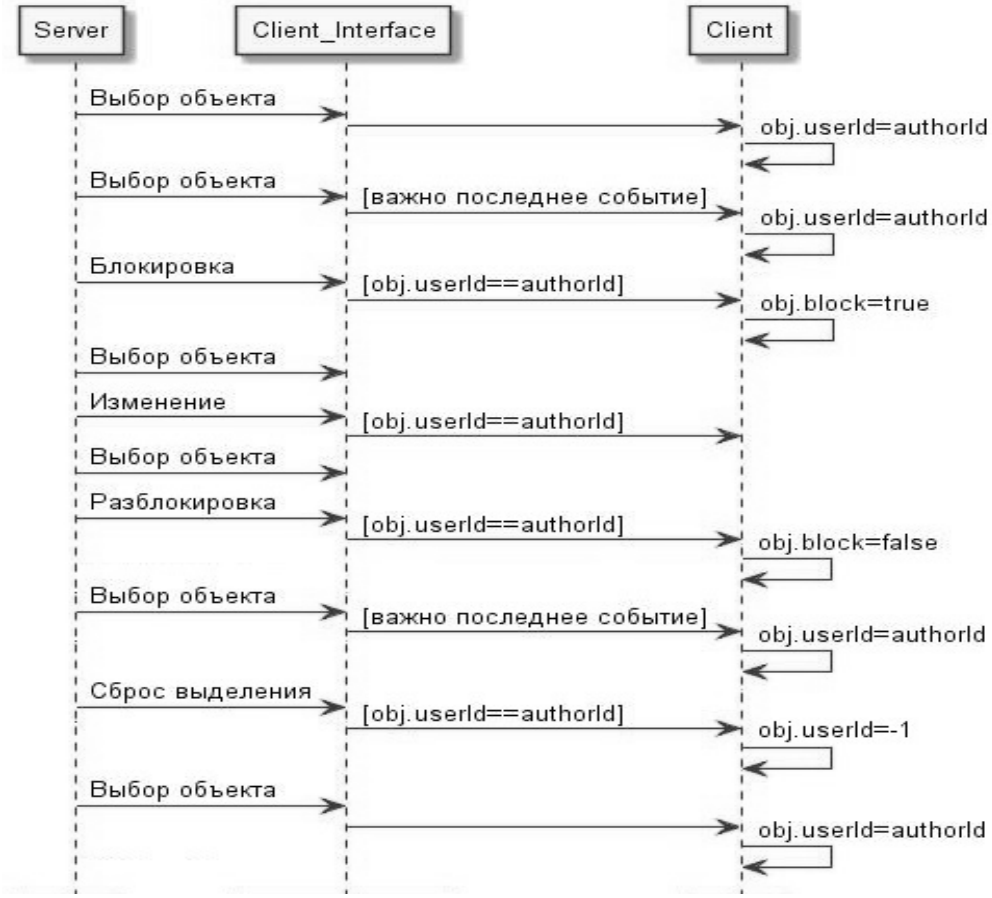

Рис. 2. Обработка событий блокирования и разблокирования ресурса клиентом – сторонним наблюдателем: obj – объект, над которым производится действие; obj.userId – идентификатор пользователя, который выбрал объект; obj.block – индикатор блокировки объекта; authorId – идентификатор автора события

Если объект заблокирован (событие блокировки было принято), запрет на его использование также действует внутри приложения.

На рис. 2 приведена последовательность обработки событий блокировки клиентом, который наблюдает происходящие события, на рис. 3 – клиентом, который является автором этих событий. На этих рисунках события блокировки и разблокировки не удалены для случая, когда для события выделения объекта важно первое событие. Наличие стрелки от Client_Interface к Client означает, что событие принимается, т.е. в приложении производятся необходимые изменения, в квадратных скобках указывается условие принятия события к исполнению, отсутствие стрелки означает отказ от выполнения события.

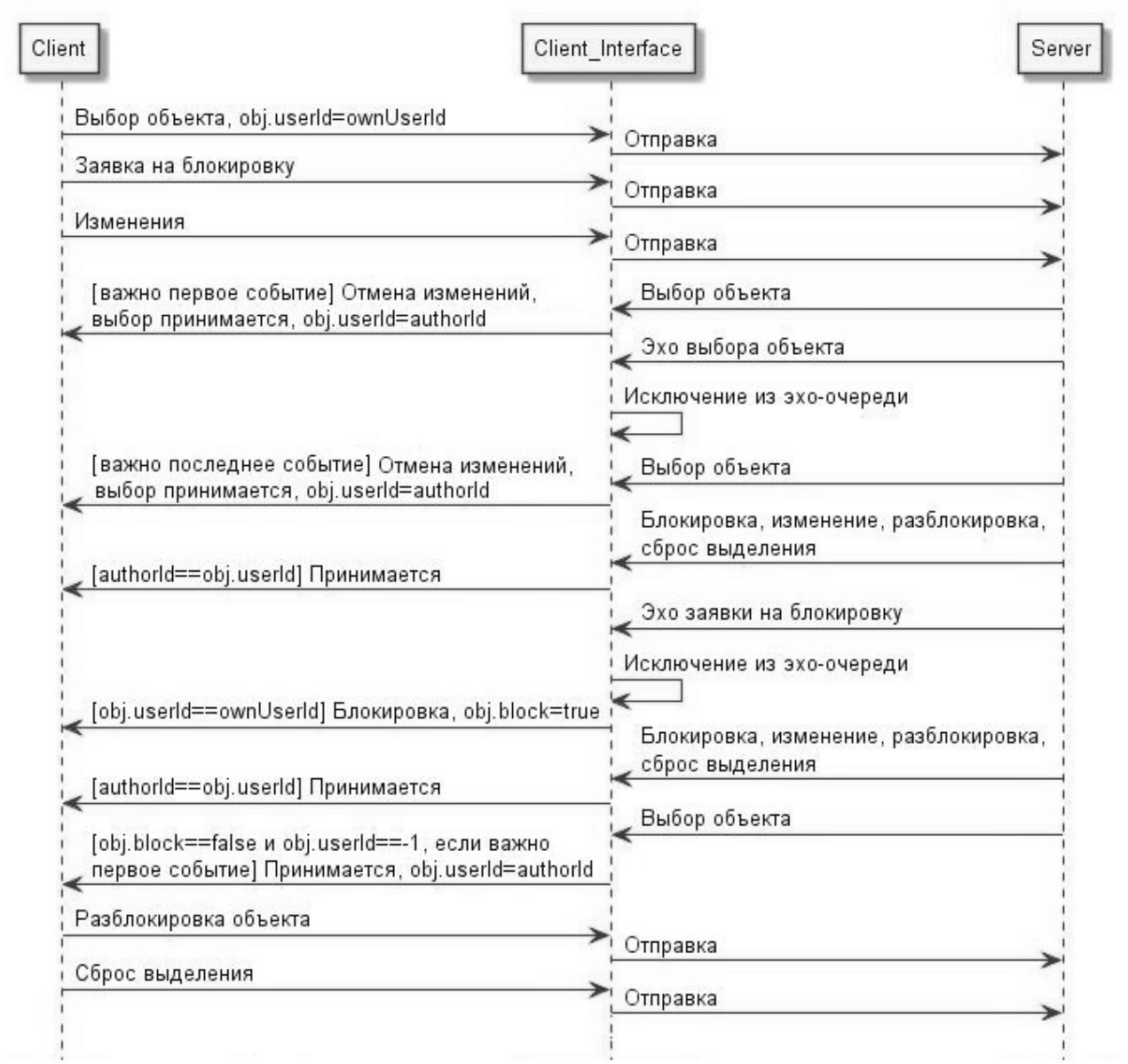

Рис. 3. Обработка событий блокирования и разблокирования ресурса клиентом – автором событий: ownUserId – идентификатор пользователя клиентского приложения. На рисунке опущено возвращение эха разблокировки объекта и добавление собственных событий в эхо-очередь

2. Сообщения синхронизации

Рассмотрим способы обмена сообщениями в синхронизируемой системе и их возможный состав. Клиент может отправлять сообщения по факту наступления события либо формировать очередь сообщений о произошедших событиях и отправлять их каждый заданный временной интервал. Для непрерывных действий (например, движения объектов), которые генерируют слишком много событий, рекомендуется использовать второй способ, поскольку большое число сообщений может затруднять связь.

Сервер может отправлять полученные сообщения по факту получения либо по запросу клиента. В последнем случае клиент может запросить версии последних изменений, чтобы не обновлять не изменившуюся с последнего запроса информацию.

Формат сообщения включает в себя название команды, идентификатор автора события, идентификатор объекта, в отношении которого произошло событие, информацию о состоянии объекта. Сервер может хранить очередь полученных сообщений, если в них указываются изменения состояния, или одно последнее сообщение, если в нем хранится состояние объекта в целом.

Для максимального исключения избыточности информации в пересылаемых сообщениях разработчик может рассматривать следующие варианты формата сообщений:

1) хранение всего состояния (всех параметров) или его части (подмножества параметров) приложения в последнем варианте;

2) хранение очереди всех поступивших изменений;

3) хранение последнего состояния для каждого вида сообщения и для каждого объекта (имя команды получается соединением названия команды и идентификатора объекта);

4) хранение очереди поступивших сообщений с регулярным сохранением состояния приложения в целом или его части.

Все указанные варианты имеют очевидную избыточность хранения информации. В первом случае предполагается сохранять состояние объектов, с которыми, возможно, никаких изменений не произошло. Во втором случае сохраняется информация, которая со временем неизбежно устаревает, например, при движении объекта сервер должен будет сохранять все нюансы этого движения. В третьем варианте предлагается генерация слишком большого числа имен сообщений. Четвертый вариант является компромиссным, он позволяет регулярно удалять большие объемы информации об изменениях, но при этом теряется возможность использовать удаленную информацию для отмены действий пользователей.

Для примера приведем порядок синхронизации содержимого виртуальных досок в виртуальном мире vAcademia [13]. Все действия (типовые команды) пользователей сохраняются на сервере в виде бинарного массива. Бинарному массиву ставится в соответствие его последняя версия. Клиенты системы регулярно запрашивают номер последней версии, и если она изменилась, то запрашивают массив всех произведенных действий и перерисовывают содержимое виртуальной доски заново.

Далее будет предполагаться, что клиент накапливает очередь сообщений и отправляет их серверу каждый интервал времени, сервер рассылает сообщения по факту поступления, данные проекта будут храниться на сервере в виде очереди поступающих сообщений с периодическим сохранением полной версии проекта.

3. Методика синхронизации online-приложений

Предлагаемая методика представляет последовательность шагов по синхронизации онлайн-приложения и предполагает наличие следующих позиций:

– отлаженный исходный код приложения;

– инструменты разработки приложения;

– инструменты разработки клиент-серверной архитектуры;

– работающее приложение сервера.

В функции серверного приложения должны входить следующие возможности:

– получение сообщений;

– отправление их участникам, включая автора сообщения;

– сохранение сообщений под указанным именем;

– отправление сохраненных сообщений вновь подключившимся участникам;

– обеспечение режима сохранения сообщения: очередь изменений либо последнее состояние.

Порядок синхронизации приложения включает следующие пункты:

– определение перечня событий, подлежащих синхронизации;

– для каждого события определяется режим синхронизации: важно первое или последнее сообщение либо блокировка (определяется событие выбора объекта);

– для каждого события разрабатывается состав полезного содержимого сообщения (не считая названия команды, идентификаторов автора и объекта);

– для каждого события определяются режим хранения на сервере (очередь изменений или последнее состояние), потребность в эхо-ответе;

– программная реализация;

– отладка и тестирование.

Синхронизируемые события могут подразделяться на следующие группы:

– создание или загрузка проекта;

– редактирование объектов;

– навигация (например, переход на стартовую страницу);

– организация (например, установка временного интервала синхронизации).

При создании или загрузке проекта до возвращения эха следует отменять все поступающие сообщения относительно предметной области приложения, чтобы не принять к исполнению команды, предназначенные для предыдущего проекта. Такие события, как сохранение проекта на стороннем сервере хранения, открытие страницы справки и другие, синхронизировать не обязательно.

4. Программная реализация

В рамках программной реализации предлагается добавить в код приложения следующие пункты (рис. 4):

– класс, отвечающий за связь с сервером (NetInterface);

– класс-журнал, отвечающий за накопление событий для отправки, обработку конфликтных ситуаций, применение полученных сообщений (Journal);

– главный класс (MainClass) приложения для каждого синхронизируемого события, предоставляющий функцию, которая вызывается из внутренних частей кода приложения и отправляет событие в журнал для регистрации;

– классы синхронизируемых объектов (SyncObject), включающие в себя параметры userId и blocked, вызовы функций регистрации в журнале событий.

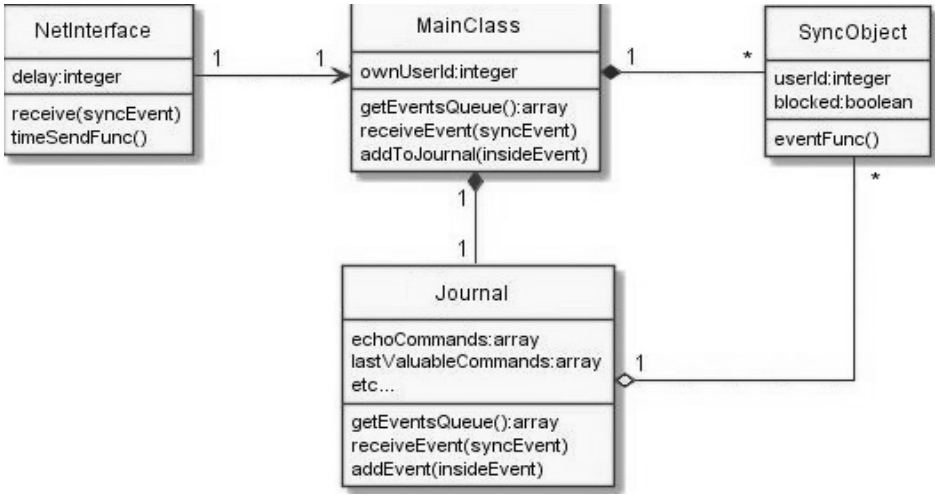

Рис. 4. Внедрение разработанных алгоритмов в код приложения

Самым сложным для реализации является класс-журнал. Журнал предоставляет функцию добавления внутреннего события, получение очереди накопленных событий (сообщений) для отправки, функцию обработки поступающих из сети событий. Непрерывные события (вроде движения) помещаются в очередь накопленных событий непосредственно перед отправлением. При добавлении события оно сохраняется в формате, удобном для отправки: сохраняется имя команды, идентификатор автора и объекта, полезная нагрузка, необходимость возвращения эха, идентификатор эха (уникальный в преде-

лах клиентского приложения), параметр сохранения информации на сервере в виде очереди изменений или последнего состояния объекта. В журнале должен быть создан массив событий (эхо-очередь), для которых ожидается возвращение эха.

Для определения вида события в журнале используются списки названий команд:

– для которых необходимо эхо;

– для которых необходимо сохранение в виде очереди на сервере;

– события создания проекта;

– события, требующие идентичности автора (блокировка, разблокировка, редактирование, сброс выделения объекта);

– события, не связанные с предметной областью (установка интервала синхронизации, переход на стартовую страницу);

– события, для которых важно последнее событие.

Функция обработки полученного из сети события должна реализовывать следующий алгоритм:

1. Если ожидается эхо создания проекта:

1.1) если вернулось эхо любого события, удалить его из эхо-очереди;

1.2) иначе если пришло событие, не связанное с предметной областью, принять согласно пунктам 4 и 5.

2. Иначе если вернулось эхо (определить по идентификатору автора):

2.1) если речь идет о блокировке либо разблокировке и userId объекта равен идентификатору автора (т.е. собственному ownUserId), принять;

2.2) удалить из эхо-очереди.

3. Иначе если пришло событие, требующее идентичности автора:

3.1) если совпал идентификатор, принять.

4. Иначе если пришло событие, для которого важно последнее событие:

4.1) если в эхо-очереди нет подобного события, принять;

4.1.1) если это событие выбора и в эхо-очереди для выбранного объекта ожидается эхо блокировки, отменить редактирование объекта.

5. Иначе если пришло событие, для которого важно первое событие:

5.1) если в эхо-очереди есть подобное событие, отменить его;

5.2) если событие не произошло (было отменено), принять событие.

Пункты следует выполнять в указанном порядке. На первом шаге важно отклонить и не обработать сообщения, которые относятся к предыдущему проекту. На втором шаге важно не обработать собственные сообщения так же, как сообщения других пользователей. Если событие требует идентичности автора, оно не заносится в список событий, для которых важно последнее событие, т.е. на шагах 4 и 5 будет распознано программой как событие, для которого важно первое событие, и должно быть обработано только на 3-м шаге. Последние два пункта можно поменять местами. Другая организация проверки типов событий может привести к росту числа операторов сравнения и усложнить процесс разработки приложения.

В журнале также должны быть определены функции, которые могут менять состояние приложения, т.е. могут «принять» событие, и другие вспомогательные параметры и функции.

5. Синхронизация flash-приложения в Adobe Connect Meeting 9

Система вебинаров Adobe Connect Meeting 9 предназначена для проведения виртуальных совещаний и, в частности, позволяет загрузку синхронизированных flash-приложений для совместного использования. Для синхронизации приложения предоставляется инструмент разработки Collaboration Builder Toolkit SDK, доступный в среде разработки Flash Builder (ранее Flex) при использовании инструмента разработки SDK 3.6. Инструмент Collaboration Builder Toolkit SDK предоставляет разные возможности, в том числе функцию отправки сообщений серверу с параметрами: название команды; содержимое сообщения; параметры, названные «дельта» (следует ли сохранять очередь поступающих сообщений или

сохранять последнее сообщение) и «эхо» (следует ли возвратить эхо сообщения). Роль сервера здесь играет сама система Adobe Connect Meeting.

Приложение также может быть разработано в среде разработки Flash Professional и экспортировано в Flash Builder при помощи инструмента Flex Component Kit, так что в Flash Builder разрабатывается код (flex-оболочка), ответственный за связь с сервером, в то время как основное приложение (далее назовем его flash-частью) должно предоставлять необходимый интерфейс для обмена информацией и включает в себя основной функционал приложения (рис. 5).

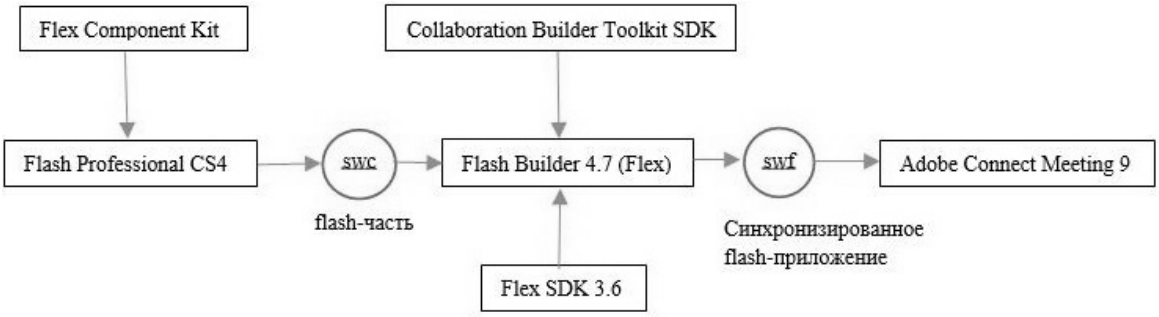

Рис. 5. Схема разработки flash-приложения, синхронизированного в системе вебинаров Adobe Connect Meeting, “swc” и “swf” – расширения выходных продуктов (файлов)

Указанная схема была применена для разработки flash-приложения «Мозговой штурм» [14]. Приложение позволяет совместно построить дерево суждений, состоящее из вершин, соединенных прямыми линиями. Каждой вершине можно назначить текст, размер, цвет и толщину обводки вершины, цвет и толщину линии к родительской вершине. Вершины можно произвольно перемещать. Вершина выделяется желтым цветом, если она выбрана самим пользователем, и красным, если она выбрана другим участником виртуального собрания (рис. 6).

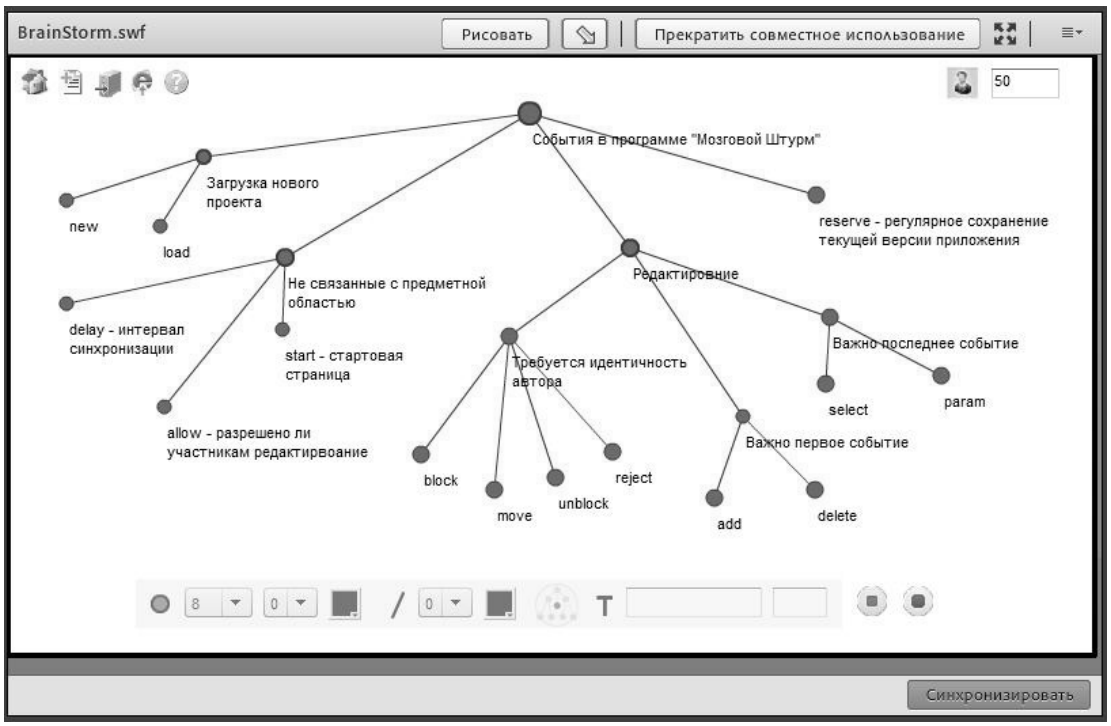

Рис. 6. Программа «Мозговой Штурм» в окне вебинара Adobe Connect Meeting 9; на рабочем поле отображены 14 используемых для синхронизации приложения событий

Очередь сообщений накапливается во flash-части приложения в ходе работы пользователя. Flex-оболочка извлекает эту очередь и отправляет серверу каждый интервал синхронизации. Событие сохра-

нения полной версии проекта (reserve) создается пользователем с наименьшим идентификатором каждые 100 интервалов синхронизации и отправляется серверу вместе с сообщениями об обнулении всех остальных событий.

Во flash-части приложения синхронизируются следующие 13 видов событий: start (переход на стартовую страницу), new (создать проект), load (загрузить проект), delay (установить интервал синхронизации), allow (разрешить участникам редактировать проект), select (выбрать вершину), param (установить параметры вершины), move (движение вершины), add (добавление вершины), delete (удаление вершины), reject (сброс выделения вершины), block (заблокировать вершину), unblock (разблокировать вершину).

События по своим характеристикам распределены следующим образом:

– не ожидающие эха: move, delete, reject;

– не хранящиеся на сервере в виде очереди: start, new, load, delay, allow;

– не относящиеся к предметной области: start, delay, allow;

– события создания проекта: new, load;

– важно последнее событие: select, param, start, delay, allow, new, load;

– важна идентичность автора: block, move, unblock, reject.

Приложение «Мозговой штурм» опубликовано в сети Интернет, с ним можно ознакомиться по адресу [15].

Заключение

В статье рассматривается синхронизация многопользовательских онлайн-приложений относительно конфликтных ситуаций, когда одно и то же событие происходит в разных клиентских приложениях одновременно. При возникновении конфликта выбирается только один пользователь, чье событие принимается как произошедшее, события остальных пользователей отменяются. В работе используются два правила разрешения конфликтов, основанные на временной последовательности поступления запросов (выбирается первый или последний пользователь), а также механизм блокировки объекта на время редактирования. При этом блокировке (и редактированию) обязательно предшествует событие выделения объекта, для которого, в свою очередь, определяется правило выбора пользователя (первый или последний). Блокировка и редактирование доступны только тому пользователю, который объект выбрал.

Указанные правила реализуются приведенными в статье алгоритмами на основе возвращаемого сервером эха собственных событий (предполагается, что используемые протоколы передачи данных обеспечивают единый порядок поступления сообщений). Для системы с клиент-серверной архитектурой, в которой события обрабатываются на стороне клиента, очередь сообщений отправляется серверу каждый интервал времени, сервер обеспечивает хранение и рассылку сообщений, приложение реализует события редактирования объектов, разработана методика синхронизации многопользовательских онлайн-приложений. В рамках этой методики предлагается вариант программной реализации алгоритмов, в частности приводится результирующий алгоритм по обработке поступающих в приложение клиента сообщений.

Разработанная методика в итоге была применена для синхронизации flash-приложения «Мозговой штурм» в системе вебинаров (т.е. виртуальных совещаний) Adobe Connect Meeting 9, предназначенного для совместного редактирования логического дерева.

ЛИТЕРАТУРА

1. Демиш В.О., Пищик Б.Н. Синхронизация данных на мобильных платформах // Вестник Новосибирского государственного университета. Сер. Информационные технологии. 2013. Т. 11, № 4. С. 46–58.

2. Макаревич А.С. Механизм защищенной синхронизации данных на базе протокола BitTorrent в распределенной вычислительной среде // Электронные средства и системы управления. 2007. № 2. С. 63–65.

3. Баранов Д.Е., Давыденко В.А., Цупранков А.В., Янишевская А.Г., Василевский В.Б., Соседко В.В. Разработка программного приложения для синхронизации базы данных справочника материалов и сортаментов // Инженерный вестник Дона. 2013. Т. 25, № 2 (25). С. 55.

4. Allen B. Downey The little book of semaphores. URL: http://greenteapress.com/semaphores/downey08semaphores.pdf (access date: 22.10.15).
5. Gropp W. Tutorial on MPI: The message-passing interface. URL: http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/talk.html (access date: 05.11.15).
6. Сетевое программирование в ОС UNIX. URL: http://www.opennet.ru/docs/RUS/tcpip/netprg0.html (access date: 05.11.15).
7. Семёнов Д.Е. Использование протоколов и их синхронизация для обмена данными в клиент-серверных приложениях // Современные техника и технологии : 13-я Междунар. науч.-практическая конференция студентов, аспирантов и молодых ученых, г. Томск, 26–30 марта 2007 г. : в 3 т. / Томский политехнический университет. Томск, 2007. Т. 2. С. 428–430.
8. Демидов А.А. Топологические методы в проектировании системы синхронизации конкурирующих транзакций распределенной базы данных // Программные системы: теория и приложения. 2011. Т. 2, № 4. С. 139–152.
9. Плутенко А.Д., Ситников А.А. Проблемы имитационного моделирования процесса журнальной репликации данных в распределенных СУБД // Информатика и системы управления. 2005. № 1. С. 3–15.
10. Демидов А.А. Об особенностях организации СУБД в MPP-системе. URL: http://psta.psiras.ru/read/psta2014_4_195-205.pdf (дата обращения: 22.10.15).
11. Калдвел Н. Устранение эффекта запаздывания с использованием кубических сплайнов. URL: http://www.gamedev.ru/code/articles/?id=4257 (дата обращения: 22.10.15).
12. Карпов А. Как создать онлайн-игру. URL: http://www.ant-karlov.ru/kak-sozdat-mnogopolzovatelskuu-igru-anons.html (дата обращения: 22.10.15).
13. Сморкалов А.Ю. Реализация инструмента графического комментирования на виртуальной доске на графических потоковых процессорах // Образовательные технологии и общество. 2012. Т. 15, № 4. С. 444–456.
14. Царева Е.В. Использование flash-технологий для расширения функциональности LMS Moodle и системы вебинаров Adobe Connect Meeting // Высшее образование сегодня. 2014. № 8. С. 30–33.
15. Мозговой штурм. URL: http://flash_services.lcg.tpu.ru/brain_storm/index.html (дата обращения: 22.10.15).

Царева Елена Владимировна. E-mail: tsareva@tpu.ru
Томский политехнический университет

Поступила в редакцию 21 августа 2015 г.

Tsareva Elena V. (Tomsk Polytechnic University, Russian Federation).
Resolution of conflicts during synchronization multiplayer online-application.
Keywords: synchronization; multiplayer application; access conflict; locking; collaborative use.

DOI: 10.17223/19988605/34/9

In information technology synchronization problem is quite urgent: the synchronization required when providing multi-tasking operating systems, in parallel and multi-threaded programming, with multi-user database. Besides synchronization process there is also the problem of synchronization of data, i.e. bringing copies of large amounts of data to one version, for example, use of cloud services, update mirroring servers or remote databases. Data synchronization can occur singly or in real time, for example, in the distributed databases and the multi-user online applications.

Task of synchronization of multiplayer online applications is different from other similar tasks. Its features are a critical time delays, reject all but one of the conflicting requests, the execution of their own actions (or cancellation of incoming events) in the application before return of a special response from the server, optionally (but preferably) lock objects on the editing, the possibility of conflict of different user selection rules for related events.

Multi-user applications can be divided into online games and applications of collaborative editing. Multiplayer games differ from collaborative editing systems by the presence of competition, conflicts arise there not only due to simultaneous access, but also different (due to time delays) developments in various client applications of system.

For conflict resolution two selection rules can be used, they based on the time sequence of receipt requests: select the first or the last user. For the authors of events the sequence is determined by the returned from the server echo events: before the return of the echo received events are considered as events occurred before, and after returning of the echo – as events occurred later. Similar conclusions are qualified because used data transfer protocols provides a unified order for receipt of messages to all customers.

For conflict resolution is also used the lock of the editing object. Selecting an object certainly precedes block event (and then edit event), for which there is determined the choice rule (first or last user). Lock, unlock, edit and deselection are only available to the user that has selected an object. Locked element is also not available for the new events of editing.

These rules are implemented in the proposed algorithms. For a system with a client-server architecture, in which events are processed on the client side, messages are sent to the server each time interval, the server provides the storage of messages, and the application implements the events editing objects, there is developed methodology of synchronizing of multi-user online applications. As part of this methodology there are proposed to list all the synchronized events, to define a policy for them for resolving the conflicts, to develop content and order of storage on the server.

For a software implementation in the application code there are proposed to add a class of network interface responsible for connection to the server, the class journal in which application events are recorded, and where network events are

processed. The main class must provide the registration function of the events in the journal. In classes of used objects there are offered to add a variable that stores the user ID, flag of the object lock, function calls of event logging, which is determined in the main class. The paper also presents the algorithm for processing of incoming messages to the client application designed for journal class.

The developed methodology was eventually used to synchronize the flash-application "Brainstorm" in the webinars (i.e., virtual meetings) Adobe Connect Meeting 9. The application allows you to build logic tree together, to determine the color, stroke width, and other parameters of vertices and connecting them lines, specify the text labels to the tops. There are 14 kinds of events that synchronized in application, including removing and adding vertices, changing settings and vertices moving, loading the project, the interval timing, and other events.

REFERENCES

1. Demish, V.O. & Pishchik, B.N. (2013) Data synchronization on mobile platforms. *Vestnik Novosibirskogo gosudarstvennogo universiteta. Ser. Informatsionnye tekhnologii – Novosibirsk State University Journal of Information Technologies*. 11(4). pp. 46-58. (In Russian).
2. Makarevich, A.S. (2007) Mekhanizm zashchishchennoy sinkhronizatsii dannykh na baze protokola BitTorrent v raspredelennoy vychislitel'noy srede [A mechanism for protected data synchronization based on BITTORRENT protocol in the distributed computer network]. *Elektronnye sredstva i sistemy upravleniya*. 2. pp. 63-65.
3. Baranov, D.E., Davydenko, V.A., Tsuprankov, A.V., Yanishevskaya, A.G., Vasilevskiy, V.B. & Sosedko, V.V. (2013) Development of software applications for database synchronization reference of materials and assortments. *Inzhenernyy vestnik Dona – Engineering Journal of Don*. 2(25). pp. 55.
4. Downey, A.B. (n.d.) *The little book of semaphores*. [Online] Available from: http://greenteapress.com/semaphores/downey08semaphores.pdf. (Accessed: 22nd October 2015).
5. Gropp, W. (n.d.) *Tutorial on MPI: The message-passing interface*. [Online] Available from: http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/talk.html. (Accessed: 5th November 2015).
6. Fedoruk, V.G. (n.d.) *Setevoe programmirovanie v OS UNIX* [Network Programming in the OS UNIX]. [Online] Available from: http://www.opennet.ru/docs/RUS/tcpip/netprg0.html. (Accessed: 5th November 2015).
7. Semenov, D.E. (2007) [Using the protocols and timing for exchanging data in a client-server applications]. *Sovremennye tekhnika i tekhnologii* [Modern technique and technilogies]. The 13th International Scientific Conference of Students, Postgraduates and Young Scientists. Tomks. 26 to 30 March, 2007. Vol. 2. Tomsk: Tomsk Polytechnic University. pp. 428-430.
8. Demidov, A.A. (2011) Topologicheskie metody v proektirovanii sistemy sinkhronizatsii konkuriruyushchikh tranzaktsiy raspredelennoy bazy dannykh [Topological methods in the synchronization system design for concurrent transactions in distributed databases]. *Programmnye sistemy: teoriya i prilozheniya*. 2(4). pp. 139-152.
9. Plutenko, A.D. & Sitnikov, A.A. (2005) Problemy imitatsionnogo modelirovaniya protsessa zhurnal'noy replikatsii dannykh v raspredelennykh SUBD [Problems of simulation of magazine data replication in distributed database]. *Informatika i sistemy upravleniya*. 1. pp. 3-15.
10. Demidov, A.A. (n.d.) *Ob osobennostyakh organizatsii SUBD v MPP-sisteme* [About the peculiarities of the organization database in the MRP-system]. [Online] Available from: http://psta.psiras.ru/read/psta2014_4_195-205.pdf. (Accessed: 22nd October 2015).
11. Caldwell, N. (2002) *Ustranenie effekta zapazdyvaniya s ispol'zovaniem kubicheskikh splaynov* [Defeating lag with cubic splines]. [Online] Available from: http://www.gamedev.ru/code/articles/?id=4257. (Accessed: 22nd October 2015).
12. Karpov, A. (n.d.) *Kak sozdat' onlayn-igru* [How to design an online game]. [Online] Available from: http://www.ant-karlov.ru/kak-sozdat-mnogopolzovatelskuu-igru-anons.html. (Accessed: 22nd October 2015).
13. Smorkalov, A.Yu. (2012) Realizatsiya instrumenta graficheskogo kommentirovaniya na virtual'noy doske na graficheskikh potokovykh protsessorakh [Tool implementation for graphic commenting on the virtual board on graphic stream processor]. *Obrazovatel'nye tekhnologii i obshchestvo*. 15(4). pp. 444-456.
14. Tsareva, E.V. (2014) The use of flash-technology for extension of LMS Moodle and webinar system Adobe Connect Meeting. *Vysshee obrazovanie segodnya – The Higher Education Today*. 8. pp. 30-33. (In Russian).
15. Tomsk Polytechnic University. (n.d.) *Mozgovoy shturm* [Brain Storm]. [Online] Available from: http://flash_services.lcg.tpu.ru/brain_storm/index.html. (Accessed: 22nd October 2015).