Представление больших наборов динамически изменяющихся удаленных данных в приложениях WPF

А. С. Чайковская, К. Я. Кудрявцев, Е. К. Павлов

***Аннотация* – Данная статья посвящена проблеме представления экстремально больших наборов динамически изменяющихся данных в приложениях WPF. В качестве основных методов повышения производительности приложения используются виртуализация данных и пользовательского интерфейса. Разрабатывается механизм виртуализации данных и алгоритмы обновления списка, обеспечивающие минимизацию нагрузки на канал связи с источником данных. Выполняется анализ возможностей представленного решения и его сравнение с существующими средствами.**

***Ключевые слова* – .Net; WPF; отображение больших наборов данных в WPF; виртуализация данных; обновление виртуализуемого списка в режиме реального времени.**

I. Введение

WPF (Windows Presentation Foundation) является графической подсистемой в рамках платформы .Net. Она предоставляет широкий спектр возможностей по созданию интерактивных настольных приложений. В частности к ним относятся элементы управления, шаблоны и стили, графика и анимация, механизмы команд и привязки данных.

В настоящее время в WPF существует множество стандартных средств для отображения данных в пользовательских приложениях. Они прекрасно подходят для решения большинства задач. Однако когда речь идет о наборах данных, содержащих более миллиона объектов, при их использовании приложение перестает справляться с такой нагрузкой и работать с ним становится невозможно.

Существует два метода повышения производительности приложения WPF при представлении больших коллекций – виртуализация пользовательского интерфейса и виртуализация данных [1]. Виртуализация пользовательского интерфейса состоит в оптимизации использования ресурсов элементами управления при отображении данных. Она поддерживается стандартными элементами управления WPF. Её использование позволяет заметно улучшить производительность приложения, но только в определенных границах. При дальнейшем увеличении числа элементов в списке производительность будет всё больше снижаться, так как программе становится всё сложнее оперировать данными, требуется много памяти для их хранения, много времени для загрузки, и, в конце концов, приложение просто перестает работать. Для решения этих проблем используется виртуализация данных.

Виртуализация данных - это механизм, позволяющий приложению размещать в памяти не весь объем имеющихся данных, а только необходимую их часть. В этом случае фактический объем памяти, используемый для хранения данных в клиентском приложении, остается небольшим и примерно постоянным при любом числе элементов в коллекции.

Так как WPF предназначена для построения пользовательского интерфейса и не имеет отношения к способам хранения и обработки данных, виртуализация данных не является встроенной в WPF функцией. Поэтому для решения этой проблемы многие сторонние разработчики предлагают свои решения.

Например, элемент управления WPF DataGrid от компании DevExpress использует виртуализацию данных, поддерживает асинхронную загрузку данных с сервера и динамические обновления списка. Однако он имеет существенные недостатки. DXGrid не дает возможности загружать наборы элементов. В результате, для получения каждого элемента списка он делает отдельный запрос на сервер, что приводит к экстремальной нагрузке на сеть и снижает скорость работы приложения. С другой стороны, несмотря на то, что элементы загружаются по одному, DXGrid не дает возможности при изменении объекта в источнике данных обновить только один элемент списка, ему необходимо выполнить перезагрузку всех полученных ранее данных.

Ещё один DataGrid, разработанный Syncfusion, обладает большим разнообразием возможностей виртуализации, но также не поддерживает загрузку

Статья получена 17.08.2017 г.

А.С. Чайковская, НИЯУ «МИФИ» (e-mail: chaikovskaia.a@gmail.com).

К.Я. Кудрявцев, К.т.н., доцент НИЯУ «МИФИ» (e-mail: const58@mail.ru).

Е.К. Павлов, НИЯУ «МИФИ» (email: pavlov.fml@yandex.ru).

набора элементов и позволяет отображать только статический список, его обновление не предусмотрено.

Другое готовое решение – DataGrid от XCeed – обеспечивает высокое быстродействие благодаря асинхронному взаимодействию с сервером, кэшированию данных и т. д., однако возможности динамического обновления данных он также не предоставляет.

Таким образом, несмотря на разнообразие существующих средств виртуализации данных в приложениях WPF, они не позволяют одновременно обеспечивать обновление списка в режиме реального времени и минимальную нагрузку на канал связи с сервисом данных.

В данной статье описывается разработка комплекса средств, позволяющих представлять экстремально большие наборы данных в приложениях WPF без потери производительности приложения. Комплекс разрабатывается с учетом требований динамического обновления списка и минимизации нагрузки на сеть.

II. Виртуализация данных

Выполнение работы начинается с разработки механизма виртуализации данных и класса коллекции, реализующей его. Описываются алгоритмы обновления списка. Затем создается элемент управления WPF, предназначенный для работы с виртуальной коллекцией. Этот элемент управления должен использовать все предоставляемые WPF возможности оптимизации представления. В заключение производится анализ производительности разработанного решения и сравнение ее с производительностью существующих средств, более всего подходящих для применения в подобных задачах.

Виртуализация данных осуществляется путем создания специальной фиктивной коллекции объектов, которая содержит только часть элементов и загружает объекты из источника данных, когда они становятся нужны элементу управления. Эта идея может быть реализована различными способами. В первую очередь, механизмы виртуализации различаются по порядку загрузки данных [2]. Для поставленной задачи больше подходит виртуализация с произвольным доступом. Её идея заключается в том, что в памяти хранится только часть данных, видимая пользователю, но взаимодействие строится так, будто загружены все данные. Индикатор полосы прокрутки всегда показывает положение отображаемой области в полном наборе элементов, что дает возможность пользователю переходить в любую область полного набора в любой момент времени.

Коллекция, осуществляющая виртуализацию данных с произвольным доступом, должна имитировать хранение всех элементов, то есть при запросе элементом управления количества элементов в списке она должна возвращать количество объектов в полной коллекции. В общих чертах способ реализации такой коллекции описан в [3]. При создании собственной коллекции рекомендуется в качестве базового использовать интерфейс IList<T>. IList представляет собой интерфейс коллекции объектов, доступ к которым осуществляется по индексу. Согласно [4], он обеспечивает наилучшую производительность в работе с ядром привязки данных, связывающим данные с элементами управления WPF. Класс коллекции также должен реализовывать интерфейс INotifyCollectionChanged. Он позволяет коллекции уведомлять внешний код, в частности, элементы управления, представляющие её данные в пользовательском интерфейсе, о добавлении и удалении элементов коллекции, а также об изменении всего списка. Эти интерфейсы обеспечивают возможность взаимодействия с разрабатываемым классом как с коллекцией. Внутри коллекции должны происходить загрузка объектов из источника данных, управление хранением уже загруженных объектов, кэшированием, обновлением данных и т. д.

Рассмотрим способ хранения данных в виртуальной коллекции. Исходный список условно разделим на страницы – фрагменты списка фиксированного размера. Когда внешний код будет запрашивать элемент коллекции, она будет загружать страницу, на которой он находится, и возвращать нужный её элемент. Эти действия выполняет индексатор коллекции, алгоритм работы которого будет описан ниже.

Помимо загрузки страниц, необходимо также выполнять удаление тех из них, которые больше не используются приложением. Для этого каждая страница будет иметь временную метку последнего обращения к ней. В определенные моменты будет вычисляться время, прошедшее с момента последнего обращения к странице. Если оно превышает некое пороговое значение, страница будет удалена из коллекции.

Наиболее оптимальное решение для хранения страниц в коллекции – размещение их в словаре Dictionary<TKey, TValue>. Этот класс реализован как хэш-таблица [5], и время выполнения операций с его элементами составляет O(1) [6]. Его структура позволяет использовать индексы страниц в качестве ключа, что гарантирует их уникальность и упрощает поиск страниц.

Так как помимо элементов страницы необходимо хранить также различную дополнительную информацию о ней, например, время последнего обращения, необходимо создать класс, представляющий страницу коллекции.

Ещё один момент, на который необходимо обратить внимание – способ размещения данных на странице. Можно было бы разместить их непосредственно в какой-либо коллекции, например, в списке, и связать с объектом страницы. Однако такой подход приводит к трудностям при работе с выделением элемента списка. Дело в том, что WPF определяет выделенный элемент на основе ссылки на объект, а не его индекса. Поэтому если выделен элемент, который ещё не загружен, то по завершении загрузки выделение будет снято.

Решением этой проблемы является использование объектов-оберток, содержащих объекты с данными как поле. Помимо обеспечения корректной работы с выделением, обертки позволяют хранить дополнительную информацию об элементах списка без изменения структуры классов с данными. Также они дают возможность сделать более совершенными механизмы обновления списка и во многих случаях при обновлении вызывать обработчики события PropertyChanged вместо CollectionChanged. Событие PropertyChanged дает более конкретную информацию о произошедших изменениях, а его обработчик, обновляющий представление списка, может быть вызван из любого потока, в отличие от обработчика события CollectionChanged, запуск которого возможен только из потока графического пользовательского интерфейса.

Разработанная схема хранения данных в виртуальной коллекции представлена на рис. 1. Диаграммы классов, представляющих страницы словаря и обёртки для данных, показаны на рис. 2.

Виртуальная коллекция должна иметь средства для взаимодействия с сервисом данных. Для упрощения сопровождения и повторного использования кода описывающие его функции должны быть отделены от реализации самой коллекции. Поэтому взаимодействие виртуальной коллекции с источником данных должно быть организовано через посредника, реализующего специальный интерфейс. В этом случае виртуальная коллекция сможет работать единым образом с любым источником данных, используя объект класса, реализующего этот интерфейс.

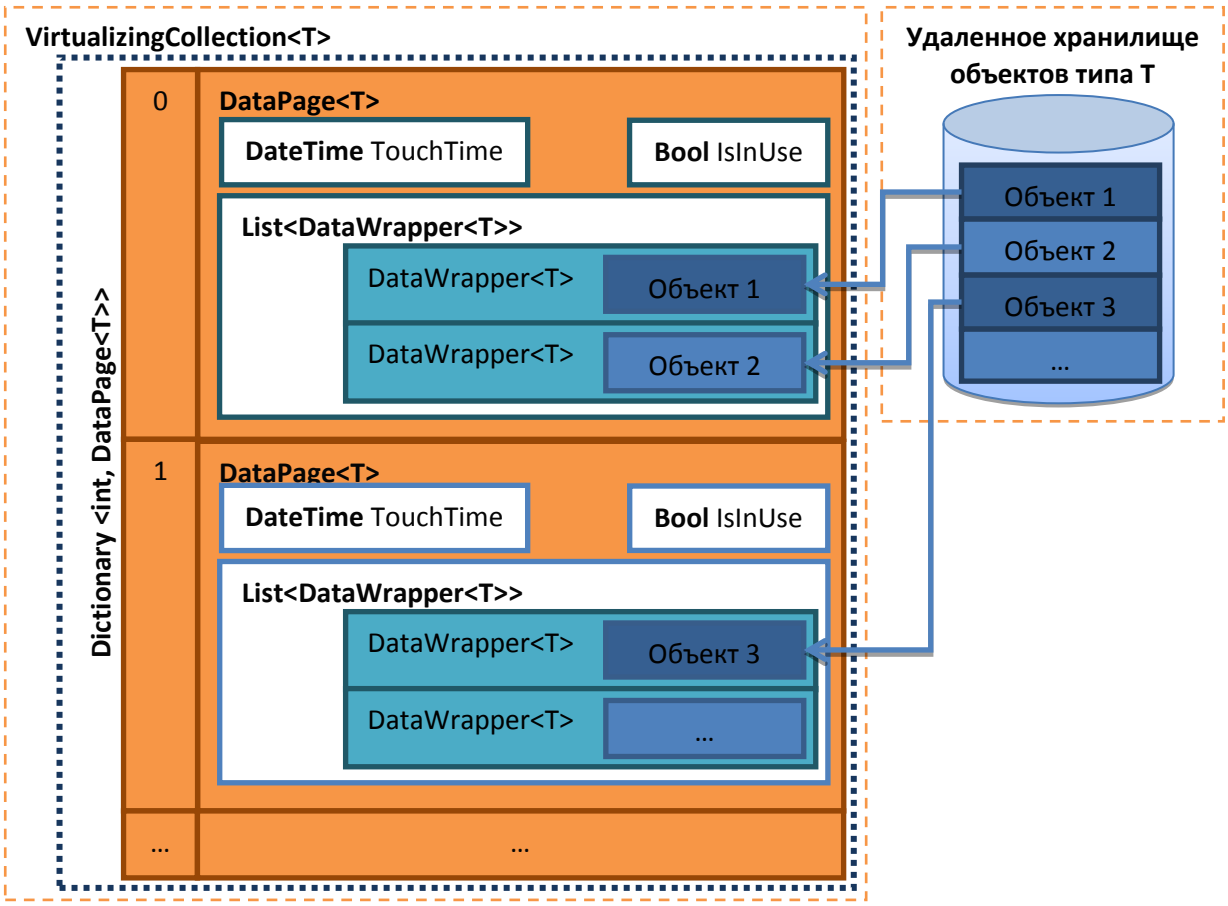

Рисунок 1 – Схема хранения объектов в виртуальной коллекции

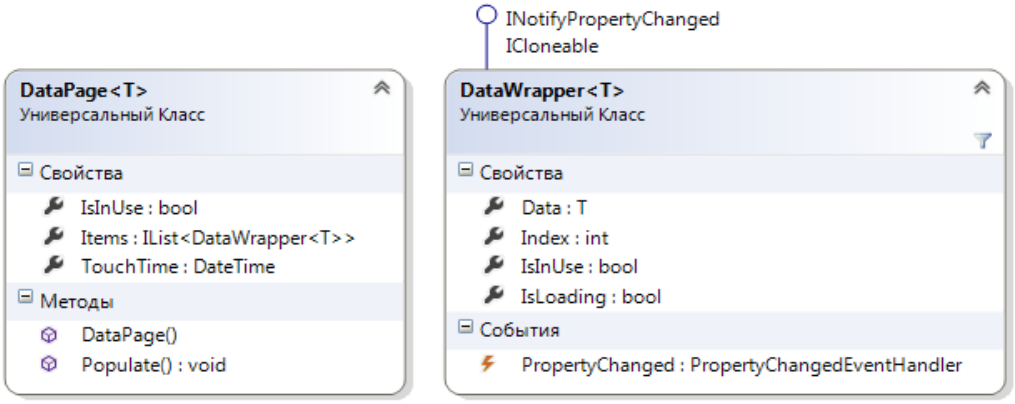

Рисунок 2 – Диаграммы классов DataPage<T> и DataWrapper<T>

III. Класс виртуальной коллекции объектов

Интерфейс поставщика данных был назван IItemsProvider. Его простейшая версия должна включать функции загрузки диапазона элементов и общего их числа в списке. Поставщик данных, реализующий этот интерфейс, будет агрегирован в класс коллекции в виде поля. В зависимости от конкретной задачи можно будет дополнить набор его функций, например, функциями фильтрации и сортировки списка.

Для обработки изменений списка поставщик данных также должен предоставлять события изменения, добавления и удаления элементов списка и обновления всего списка. Эти события будут выделены в интерфейс IServerListCallBackBehaviour.

В поведении элементов управления WPF при работе с выделением в списке существует одна особенность. Когда элемент управления получает оповещение о том, что объект был добавлен или удален из коллекции, он пытается найти выделенный ранее элемент списка, используя сохраненную ссылку на него. Эта ссылка сравнивается с каждым элементом коллекции по порядку, до тех пор, пока не будет найден соответствующий ей объект, или до достижения конца списка. При использовании виртуализации данных это означает, что при этом может быть загружено огромное количество страниц виртуальной коллекции, которые на самом деле не нужны.

В связи с этим возникает необходимость перед обновлением списка снимать выделение и восстанавливать его по завершении обновления. Так как элемент управления ничего не знает о том, началась ли процедура обновления коллекции, а виртуальная коллекция в свою очередь не имеет прямого доступа к элементу управления, нужен специальный интерфейс для их взаимодействия. Он должен содержать события, инициируемые коллекцией, в ответ на которые элемент управления будет соответствующим образом управлять выделением. Этот интерфейс был назван ISelectionManager.

Выполнения операций загрузки данных и обновления списка должны производиться в отдельном потоке, не блокируя поток пользовательского интерфейса. Поэтому будет создан класс-наследник виртуальной коллекции, осуществляющий эти действия асинхронно. Для того чтобы проинформировать пользователя о том, что коллекция загружает данные, будет использован индикатор загрузки RadBusyIndicator, разработанный компанией Telerik. Состояние индикатора будет привязано к свойству коллекции, показывающему, находится ли видимая область списка в стадии загрузки. Это свойство объявлено в интерфейсе IBusyContent.

Основные члены класса виртуальной коллекции представлены на диаграмме на рис. 3.

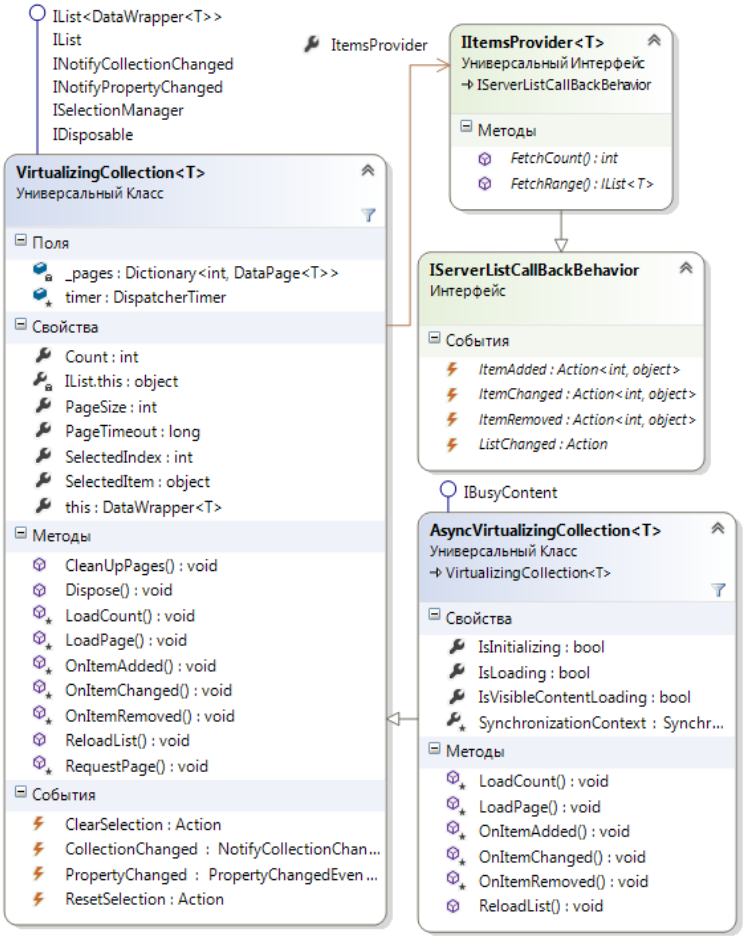

Рисунок 3 – Диаграмма классов виртуальной коллекции

Рассмотрим процессы, происходящие внутри виртуальной коллекции.

A. *Передача элемента коллекции*

При обращении к объектам коллекции внешний код использует её индексатор. Индексатор рассчитывает номер страницы, на которой находится запрошенный элемент, и его смещение в ней, и запрашивает требуемую страницу. Для большей плавности прокрутки списка также следующая или предыдущая в зависимости от положения запрошенного элемента на странице. Затем индексатор выполняет очистку памяти, занятой устаревшими страницами. После совершения всех действий, если запрошенный элемент уже загружен, индексатор передает его вызывающему коду; иначе передается элемент по умолчанию.

B. *Обращение к страницам коллекции*

Запрос страницы индексатором порождает следующую последовательность действий. Если запрашиваемая индексатором страница уже загружена в виртуальную коллекцию и присутствует в словаре, виртуальная коллекция обновляет время последнего обращения к этой странице. Иначе в словарь добавляется новая страница с нужным числом пустых оберток для данных. Затем коллекция запрашивает данные у поставщика данных и подставляет полученные данные в объекты-обертки.

C. *Очистка устаревших страниц*

При реализации механизмов очистки словарей важно обратить внимание на то, что с некоторыми объектами списка элемент управления WPF взаимодействует особо. В их число входят объекты из видимой области списка и первый элемент списка. Если удалить страницу, на которой расположен один из этих объектов, она в тот же момент будет снова запрошена элементом управления. Поэтому для сокращения числа запросов к источнику данных никогда не должны удаляться первая страница и видимая страница. Для определения видимости страницы достаточно проверить наличие подписчиков на событие PropertyChanged её элементов.

Очистка словарей будет происходить в двух случаях – перед внесением изменений в список и по таймеру. Это сочетание позволит, во-первых, сократить объем вычислений при обработке изменений, и, во-вторых, обеспечить корректную работу виртуальной коллекции со статичным списком. Удаление страницы из словаря будет производиться, если время, прошедшее с момента последнего обращения к ней, превысило заданный предел.

D. *Обновление данных*

При изменении данных поставщик данных должен инициировать одно из событий ItemAdded, ItemChanged, ItemRemoved или ListChanged. Каждому из них соответствует свой обработчик в классе виртуальной коллекции. При возникновении события обработчику будет передан измененный элемент и его индекс в списке. Последующая обработка списка будет производиться на клиентской стороне. Так как в каждый момент времени в памяти хранится лишь небольшое количество страниц коллекции, их обработка не дает заметной нагрузки на приложение, а число запросов к источнику данных, приходящихся на каждое изменение списка, будет значительно снижено в сравнении с существующими решениями, предполагающими полную перезагрузку всех страниц.

Обработка события изменения элемента тривиальна – данные внутри соответствующего объекта-обертки заменяются новыми, переданными обработчику в качестве параметра. В результате, при изменении элемента будет загружен всего один объект.

При добавлении элемента коллекция перебирает все загруженные страницы. Если все элементы страницы находятся в списке выше добавленного, она не изменяется. Если элемент добавляется в текущую страницу, он помещается в соответствующий ему объект-обертку, а данные из этой обертки и всех последующих сдвигаются вниз по списку. В случае, когда все элементы страницы находятся ниже добавленного, коллекция запрашивает новый первый элемент страницы у поставщика данных, размещает его в первой обертке и сдвигает вниз все остальные элементы страницы. Если обрабатываемая страница последняя в списке и её размер меньше установленного размера страниц в коллекции, перед сдвигом в её конец добавляется новый объект-обёртка.

Удаление элементов обрабатывается схожим образом. Отличие состоит в том, что сдвиг элементов происходит в обратном направлении и дополнительно загружаются не первые, а последние элементы страниц. Соответственно, если страница последняя, из неё исключается последний объект-обертка.

IV. Элемент управления для работы с виртуальной коллекцией

Для работы с виртуальной коллекцией был разработан новый элемент управления WPF VirtualizationListViewControl. Его структура представлена на рис. 4. Содержимым VirtualizationListViewControl является стандартный элемент управления ListView. Его настройка из внешнего кода происходит через свойства VirtualizationListViewControl. Свойство ItemsSource представляет коллекцию элементов списка, которой может являться виртуальная коллекция. Свойство SelectedItem позволяет устанавливать выделенный элемент списка. Закрытые методы ClearSelection() и ResetSelection() являются обработчиками событий снятия и восстановления выделения в списке, инициируемых виртуальной коллекцией.

Данные внутри ListView в данном решении представляются в режиме GridView – многоколоночного

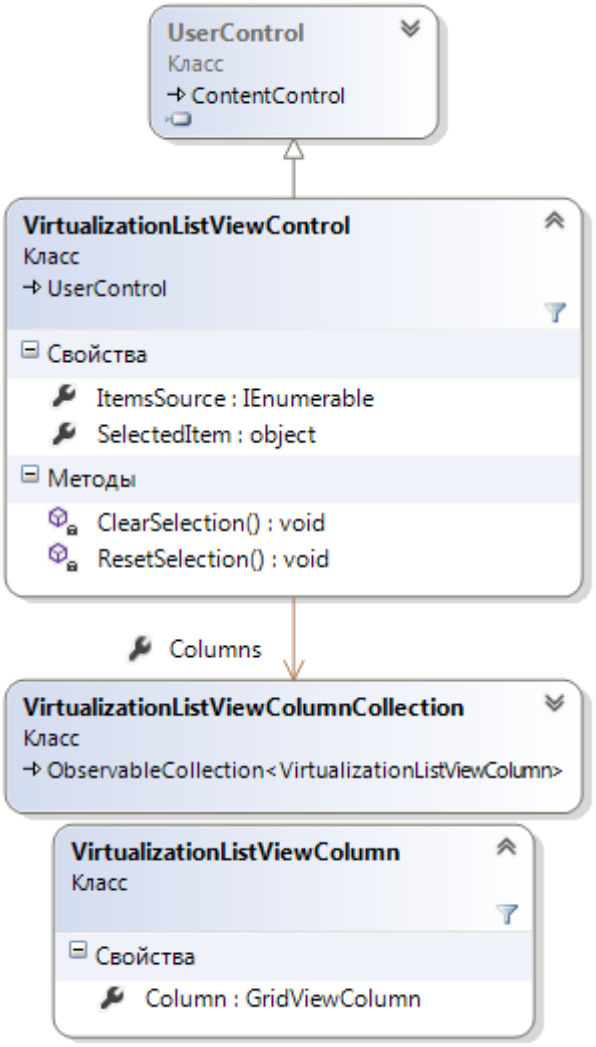

Рисунок 4 – Диаграмма классов для VirtualizationListViewControl

списка. Коллекция столбцов для GridView задается через свойство Columns VirtualizationListViewControl. Имеется возможность задать не только набор столбцов, но и шаблоны представления для данных внутри них. Также позволяется задавать различные шаблоны для объектов разных типов.

ListView настроен таким образом, чтобы он использовал все возможные средства повышения производительности пользовательского интерфейса. Рассмотрим их подробнее.

При отображении объектов WPF создает для них контейнеры разметки. Если к элементу управления привязать коллекцию объектов в качестве источника элементов, контейнер разметки будет создан для каждого объекта в отдельности. При экстремально большом размере коллекции это ощутимо сказывается на производительности приложения. При использовании виртуализации пользовательского интерфейса контейнеры разметки создаются только для элементов из видимой области. ListView по умолчанию выполняет виртуализацию интерфейса, однако это поведение может быть неявно отключено [7]. Например, такая ситуация возникает при помещении ListView внутрь области прокрутки. При решении задачи ситуации, приводящие к отключению виртуализации пользовательского интерфейса, были исключены.

Кроме того, ListView дает возможность повторного использования контейнеров разметки. Она позволяет использовать для представления объектов один и тот же набор контейнеров вместо того, чтобы удалять контейнеры объектов, вышедших из видимой области при прокрутке, и создавать новые для вошедших в нее.

Наконец, ListView имеет функцию отложенной прокрутки. Она заключается в том, что при перетаскивании ползунка прокрутки список обновляется не постоянно, а только когда пользователь отпускает ползунок. Эта возможность особенно полезна при виртуализации данных, так как иначе происходила бы загрузка всех элементов, лежащих между начальной и конечной позициями индикатора прокрутки.

Итак, мы имеем коллекцию, позволяющую выполнять виртуализацию данных, и специальный элемент управления, предназначенный для работы с ней. Для использования этих средств в приложении достаточно реализовать интерфейс поставщика данных соответствующим конкретной задаче образом. Совокупность этих компонентов обеспечивает решение поставленной в данной статье проблемы.

V. Тестирование и сравнение с существующими средствами

Сравним возможности разработанного решения с наиболее приближенным к нему по функциям коммерческим продуктом DXGrid. Как было сказано во вводной части статьи, DXGrid поддерживает только поэлементную загрузку данных. Пусть размер страницы коллекции равен 20. Тогда при её загрузке DXGrid выполнит 20 запросов к удаленному сервису данных. Описанное здесь решение выполнит загрузку за один запрос. Однако наиболее значительная разница наблюдается при обновлениях списка. Пусть в памяти хранится 2 страницы коллекции. Тогда при любом изменении списка DXGrid выполнит 40 запросов к источнику данных. Разработанный комплекс в самом худшем случае потребует 2 запроса, то-есть разница во времени будет составлять примерно 76 задержек передачи данных по сети. Однако необходимо также учесть время, затрачиваемое виртуальной коллекцией на обновление списка. Для измерения этих значений было создано демонстрационное приложение. Эксперимент проводится при отсутствии задержки передачи данных. При проведении измерений в видимой области находится начало списка, в память загружено две страницы. Начальное число элементов - 10000. Полученные значения приведены в таблице 1.

Даже если взять за основу максимальное из значений в таблице – 13 мс – получим, что DXGrid обеспечит более высокую производительность в рассматриваемой ситуации только в случае, если задержка передачи

Таблица 1 – Длительность обновления данных AsyncVirtualizingCollection<T> при изменениях списка, в мс

	Индекс изменения в списке	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Среднее**
Добавление	0	8	11	4	5	5	4	6	6
Изменение	0	1	0	0	0	3	0	0	1
Удаление	0	8	6	10	10	4	6	7	7
Добавление	10	5	5	6	5	6	4	2	5
Изменение	10	3	0	0	0	0	0	3	1
Удаление	10	6	7	10	13	11	9	9	9
Добавление	10000	6	4	1	4	4	3	5	4
Изменение	10000	1	0	2	0	0	0	0	0
Удаление	10000	3	7	4	3	4	1	2	3

данных по сети составляет менее 0,17 мс. Если же необходимо обновить больше двух страниц, то задержка должна быть ещё меньше. И даже в этой ситуации, хотя необходимое быстродействие приложение и будет обеспечено, нагрузка на сеть останется достаточно существенной. Так как в большинстве случаев задержка передачи данных больше приведенного значения, можно с уверенностью сказать, что разработанные механизмы обновления списка обеспечивают значительное повышение быстродействия приложения и снижение нагрузки на канал связи.

VI. Выводы

Описанное в данной статье решение является универсальным средством для отображения экстремально больших наборов динамически изменяющихся данных в приложениях WPF без потери производительности приложения. Оно способно работать с совершенно любым источником данных, а в случае его удаленности обеспечивает минимальную нагрузку на канал связи. Характеристики решения значительно отличаются от характеристик схожих существующих средств в лучшую сторону.

БИБЛИОГРАФИЯ

[1] Matthew MacDonald. Pro WPF 4.5 in C# / Нью Йорк,, APRESS, 2012. 1078 с.

[2] Использование виртуализации в списке или сетке (XAML). URL: https://msdn.microsoft.com/ru-ru/library/windows/apps/xaml/hh780657.aspx

[3] Mark LeBLanc. ListView and GridView data virtualization. URL: https://msdn.microsoft.com/en-us/windows/uwp/debug-test-perf/listview-and-gridview-data-optimization

[4] Оптимизация производительности: привязка данных. URL: https://msdn.microsoft.com/ru-ru/library/bb613546(v=vs.110).aspx#Binding_to_an_ItemsSource

[5] Dictionary<TKey, TValue> - класс. URL: https://msdn.microsoft.com/ru-ru/library/xfhwa508(v=vs.110).aspx

[6] C#/.NET Fundamentals: Choosing the Right Collection Class. URL:

http://geekswithblogs.net/BlackRabbitCoder/archive/2011/06/16/c.net-fundamentals-choosing-the-right-collection-class.aspx

[7] Optimizing Performance: Controls. URL: https://msdn.microsoft.com/ru-ru/library/cc716879(v=vs.110).aspx

Displaying large sets of dynamically changing remote data in WPF applications

A. S. Chajkovskaja, K. Ya. Kudrjavtsev, E. K. Pavlov

Abstract – The former article is dedicated to the problem of displaying large sets of dynamically changing remote data in WPF applications. Data and UI virtualization technologies as main tools for performance increase are described. Special data virtualization and updating algorithms are designed to minimize load on channel for communications with data source. Offered solution is analyzed and compared with other existing ones.

Keywords – .Net; WPF; displaying large data sets in WPF; data virtualization; virtualized list real-time updating.