Еще один подход к проектированию обратимых операций в программном обеспечении

М.М. Гумеров
ООО «МАСС МО»
Москва, Россия
e-mail: mgumerov@gmail.com

Аннотация[1]

Статья подвергает пересмотру и дополнению более раннюю работу автора [1]. Предлагается новый комплекс мер, позволяющий при разработке программного обеспечения во многих случаях добиться аналогичного результата, но с меньшими жертвами в отношении читаемости исходного кода и отхода от обычного стиля работы с объектами.

1. Введение

В своей диссертации [1] автор исследует некоторые вопросы проектирования механизмов отмены выполняемых операций в программном обеспечении. Поскольку комфортная работа с современными приложениями для редактирования каких-либо видов документов немыслима без возможности отмены ошибочных выполненных пользователем действий, а существующие опубликованные исследования обходили вниманием конкретную проблему, рассмотренную в диссертации, исследование в этом направлении было сочтено перспективным. Предлагаемый автором подход к проектированию включает использование неких идентификаторов для связывания тех объектов, в которые отображаются элементы документа в памяти работающего приложения.

Как показали позднейшие исследования автора, возможно более точно выделить те свойства предложенного метода, которые позволяют решить проблему связей между объектами, и применять в проектировании именно их, не следуя этому методу полностью. В данной работе будет представлена система ограничений и правил проектирования, следование которой позволяет решить рассмотренную в [1] задачу иным способом.

2. Суть задачи и раннего решения

Автор предполагает, что объектно-ориентированная программа работает с содержимым документа не непосредственно, а через некоторое объектно-ориентированное представление – комплекс объектов в памяти. Элементы, из которых состоит документ, могут иметь связи друг с другом, а также на них могут ссылаться другие объекты программы – например, хранящие информацию о проделанных операциях для обеспечения возможности отмены последних.

Суть проблемы – в следующем. Если элементы документа отображаются в объекты в оперативной памяти приложения, и связи между элементами отображаются в ссылки одних объектов на другие, и над каким-то элементом выполняется операция удаления, нужно каким-то образом удалить ссылки других объектов на соответствующий ему объект; аналогично при отмене операции удаления нужно установить этим объектам ссылки на восстановленный объект. Добиться этого непросто.

В диссертации [1] констатируется отсутствие опубликованных научных исследований обратимых операций. В то время как задача проектирования системы поддержки отмены абстрактных операций в приложениях рассматривается как научная в немалом числе исследований, причем уже в 2000-е годы ([4,5] и даже, например, [6]), задачи проектирования отдельных типов таких операций с целью обеспечения их обратимости традиционно считаются инженерными, в результате чего всякий раз для всякой операции решаются заново ([2,3]).

Автором же в [1] предпринята попытка проанализировать обобщенную задачу проектирования таких операций как научную, показана исключительность операции удаления элемента документа в приложениях-редакторах документов, и предложен был опять-таки обобщенный подход к проектированию операций удаления в различных приложениях.

В качестве решения было предложено отказаться от традиционного представления ссылок на объекты (указатели на их расположение в оперативной памяти, или ссылки .Net и Java без предположений об их внутреннем представлении), а вместо этого идентифицировать объекты неким значением-ключом. Это позволяет преодолеть отмеченные в [1] трудности восстановления объектов и связей между ними – например, если связи представлены указателями, и объект при отмене операции удаления

[1]**Труды второй международной конференции "Интеллектуальные технологии обработки информации и управления", 10 - 12 ноября, Уфа, Россия, 2014**

восстанавливается уже в другом месте оперативной памяти, не там, где находился до удаления.

В то же время, сам вывод о существовании таких трудностей опирается на ряд предположений.

1. Любой объект, на который делается ссылка, может быть уничтожен или перемещен в любой момент.
2. Состояние приложения в оперативной памяти включает в себя все содержимое обрабатываемого документа, отображенное на какие-то объекты (речь идет об объектах ООП).
3. Все участки программы, имеющие ссылку на объект, описывающий элемент документа, заинтересованы в том, чтобы ссылка оставалась неизменной при изменениях документа; если элемент изменяется, это должно приводить к соответствующему изменению внутри этого объекта, а не к переключению ссылки на какой-то другой объект, являющийся «более актуальной» проекцией этого же элемента.

Дальнейший же проведенный автором анализ показывает, что эти предположения не являются чем-то неоспоримым. Рассмотрение возможностей некоторых существующих систем объектно-реляционного отображения (ORM), таких, как Hibernate, и, что важнее, практики их использования в разных проектах, и в том числе и в Web-приложениях, позволил прийти к выводу, что, во-первых, хранение документа целиком в памяти программы не всегда является хорошей идеей, во-вторых, отказ от этого приводит к тому, что многие элементы документа не присутствуют в памяти программы постоянно, а доступны лишь в какие-то моменты времени. Например, автору известны сценарии, в которых всякий раз, как программе требуется доступ к какому-то элементу документа, для этой части заново создается объект в памяти и заполняется содержимым этого элемента; известны и другие сценарии, когда единожды созданные объекты кэшируются и могут использоваться повторно до тех пор, пока кэш не переполнится, что приведет к вытеснению из него части объектов.

Нетрудно видеть, что если всякий раз при необходимости доступа к неким данным для них создается новое объектное отображение в память, а в остальное время этих объектов в памяти нет – то в хранении информации о связях между элементами документа именно в оперативной памяти большую часть времени вообще нет необходимости. Вместо этого связи можно представлять в памяти каким-либо образом тоже именно в тот момент, когда соответствующие объекты отображаются в память.

Что до первого предположения, оно тоже не вполне верно. Так, скорее всего, объекты удаляются в процессе выполнения конкретных операций. Если некая операция не связана с удалением элементов документа, то вполне возможно ввести контракт, по которому в ходе этой операции никакие объекты не удаляются. А значит, как минимум, можно выделить некоторые периоды в работе программы, когда нет опасности удаления объектов, что снимает необходимость отказа от указателей в эти периоды.

3. Новое предложение

Новый подход к решению можно сформулировать так: ссылки между объектами можно по-прежнему представлять в виде классических указателей, но при этом всегда следует понимать, что полученная ссылка указывает на объект, который затем может быть уничтожен. Но тогда нужно дать более четкое определение понятию «затем», иначе это правило не лучше неопределенности. Предлагается такое: поскольку история изменений документа состоит из отдельных операций над ним, можно принять, что объекты гарантированно существуют до конца операции, но после окончания операции могут быть удалены и, возможно, заново прочитаны из БД уже в другой участок памяти. А могут и остаться в кэше и использоваться повторно.

Возможность ссылаться на объекты в ходе выполнения каких-то операций позволяет, например, без дополнительных ухищрений реализовать классическую для .Net привязку (binding) отдельных элементов управления из пользовательского интерфейса конкретной операции к каким-то свойствам объектов, которыми она оперирует. Конечно, можно сделать такое и с использованием ключей вместо указателей, но это означало бы дополнительное усложнение исходного кода. А при использовании указателей возможен такой способ подписки на события, который обычен для данного языка (например, механизм событий .Net Framework).

Например, имеется объект, с которым в связи «многие к одному» состоят несколько дочерних объектов; пользовательский интерфейс для некой операции редактирования предоставляет возможность добавить дочерний объект или изменить свойства родительского. При этом среди свойств родительского объекта и дочерних есть такое, как «цвет». Если задать «цвет» родительского объекта, изменяется цвет всех дочерних; если задать «цвет» дочернего, отличный от цвета остальных, «цвет» родительского делается «неопределенным». Один из способов организовать такой интерфейс – подписаться к каждому из объектов на уведомления об изменении его свойств. Если свойства объектов отображаются в каких-то элементах управления, эти элементы тоже нужно обновить. В .NET Windows Forms связать элементы с данными можно при помощи Data Binging, и делать это удобно, имея возможность ссылаться на объекты при помощи обычных ссылок .Net.

Сопутствующая проблема – отслеживание изменений объектов – в этом случае также нуждается в анализе. Здесь помогает тот факт, что каждая операция подвергает изменениям определенный известный ей

перечень объектов. Но тем не менее, ведь сама операция не знает логику, по которой одни объекты зависят от других. Поэтому, если для зависящих элементов документа не созданы объекты, вызов какой-то обработки при таких событиях становится проблемой, которую также следует решать. Тем не менее, в первом приближении будем считать, что элементы в документе связаны просто для отражения каких-то структурных связей, а не для того, чтобы каскадировать (to cascade) изменения одних элементов на какие-то другие. В тех же случаях, когда этого недостаточно, можно, хотя это, вероятно, и не лучшие варианты, либо возложить запуск такой обработки на реализацию документа, либо на саму выполняемую операцию.

Если же почему-то есть необходимость иметь ссылку на элемент документа, которая переживет рамки одной операции, тогда уже следует ссылаться по ключу.

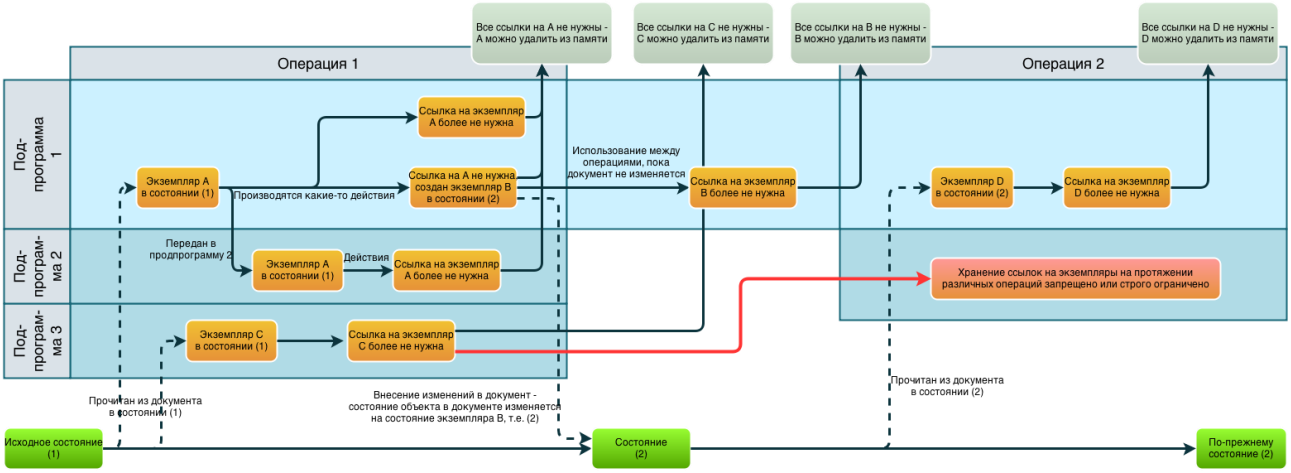

Рис. 1. Жизненный цикл ссылок на объекты, представляющие элементы документа.

Следующий шаг заключается в осознании того, что между операциями приложение не простаивает – просто оно работает в read-only режиме. Могут переключаться режимы просмотра, изменяться просматриваемая область документа, обновляться содержимое каких-то представлений. Это означает, что между операциями возможно также выполнение кода, который будет запрашивать фрагменты документа и оперировать соответствующими объектами. И во многих случаях здесь кэширование уже созданных объектов (или результатов какой-то их обработки) просто необходимо.

В действительности ничего не мешает разрешить такое кэширование (как вне операций, так и внутри них). Только двух правил следует придерживаться: 1) обеспечить способ, которым пользователь кэша узнает, что кэш больше не соответствует данным документа, 2) кэш должен хранить копию данных и, если в используемом языке программирования существует понятие владельца в смысле кого-то, кто отвечает за удаление данного объекта, то владельцем кэша должен быть тот, кто его завел для себя (т.е. сам для себя сделал копию и когда не нужна более – удалил). При необходимости созданная копия может подаваться сторонним расчетным функциям – в т.ч. и тем, которые в другой ситуации получают читаемые непосредственно из документа объекты. Откуда вытекает еще одно соглашение: если на вход какого-то метода подается объект, метод должен понимать, что этот объект может быть лишь копией «для внутреннего пользования» - или, равным образом, это может быть не внутренняя копия, а созданный на время операции объект. Т.е. никакой метод не должен делать предположений о том, как возник пришедший ему на вход объект.

Таким образом, предлагается методика проектирования, состоящая в применении следующих правил.

1. Если указатель на объект получен в некоторой операции, то в пределах операции можно использовать этот указатель.
2. Если указатель на объект получен вне операций, то как минимум до начала следующей операции можно продолжать его использовать. Либо – до того момента, пока каким-то способом не станет известно, что ссылка устарела.
3. Когда использование указателей для того, чтобы ссылаться на элементы документа в соответствии с пп. 1-2, становится более недопустимым, надлежит использовать другие механизмы – например, ранее предложенные автором в [1].
4. Допускается создание полных или частичных копий объектов, прочитанных из документа. Ответственность за жизненный цикл копии должен нести тот потребитель в программе, который их создал. Любой код, оперирующий объектами, которые могут отражать элементы документа, должен быть готов к тому, что ему могут подать на вход как объект, созданный из элемента, так и его копию, и в идеале не должен пытаться выяснить, что из этого является истиной.

Следование этой методике позволяет добиться актуальности объектов, на которые отображается

документ, и их связей во многих случаях даже без применения таких жестких мер, которые были предложены в [1].

4. Заключение

1. Выявлен комплекс допущений, обсуславливающий трудности проектирования отменяемой операции удаления, констатированные в [1].
2. Предложен подход к проектированию, позволяющий избежать возникновения этих трудностей, и в то же время по-прежнему абстрагированный от специфики конкретных приложений.
3. При этом не проводилось формальное доказательство того, что данный подход применим во всех ситуациях, в которых применимы результаты [1].

Список используемых источников

1. Гумеров М. М. Проектирование обратимых операций над объектами на основе шаблонов при разработке программного обеспечения.: дис… канд. техн. наук: 05.13.11: защищена 25.11.2011: утв. 26.04.2012 — Уфа., 2011. — 118 с.
2. Пользователь DRogov. Технический отчет сотрудников Developer Express: «Undo/Redo — Хвост виляет собакой» [Электронный ресурс]. URL: http://habrahabr.ru/company/devexpress/blog/104167/ (дата обращения 24.04.2014) [User DRogov. “Developer Express: Technical Report «Undo/Redo: Wagging The Dog»”]
3. Zhang M., Wang K., “Implementing Undo/Redo in PDF Studio Using Object-Oriented Design Pattern”, in Proc. TOOLS-Asia'00, 2000, p.58
4. Choudhary R., Dewan P. “A general multi-user undo/redo model”, in Proc. ECSCW, 1995, Stockholm, pp. 231-246
5. Berlage T., Genau A. “From Undo to Multi-User Applications”, in Proc. VCHCI '93, Sept. 1993, pp. 213-124
6. Bernstein P.A., Hadzilacos V., Goodman N. “Concurrency Control and Recovery in Database Systems”. Addison Wesley, 1987, 370 p., ISBN 9780201107159