ПРИМЕНЕНИЕ МЕТОДОВ И СРЕДСТВ МОДЕЛИРОВАНИЯ

УДК 004.5

А.Л. Масюк, аспирант
Донецкий национальный технический университет «ДонНТУ»
(Украина, 85300, Покровск, пл. Шибанкова, 2,
e-mail: ars.masiuk@gmail.com)

Метод Diff для имплементации стека отмены недавних действий пользователя

Рассмотрены наиболее популярные в настоящее время паттерны имплементации стека отмены недавних действий Memento и Command, проанализированы их достоинства и недостатки. Предложен метод Diff, в котором сочетаются преимущества обозначенных паттернов с простотой и универсальностью реализации. Данный метод проверен на практике при разработке визуального редактора графов шахтных вентиляционных сетей и их параметров, а также в других подобных приложениях. Доказана его эффективность по таким критериям, как быстродействие, расход памяти, трудозатраты на реализацию.

Розглянуто найпопулярніші на даний час патерни імплементації стеку відміни нещодавніх дій Memento та Command, проаналізовано їх позитивні та негативні властивості. Запропоновано метод Diff, в якому поєднано переваги означених патернів та простота і універсальність реалізації. Даний метод практично перевірено під час розробки візуального редактора графів шахтних вентиляційних мереж та їх параметрів, а також в інших подібних програмах. Доведено його ефективність за такими критеріями, як швидкодія, споживання пам'яті, витрати на реалізацію.

Ключевые слова: визуальное редактирование, стек отмены, графический интерфейс.

В большинстве современных программ, реализующих идеологию визуального редактирования моделей данных, используется концепция стека отмены недавних действий пользователя. Это значительно облегчает редактирование, так как с помощью такого стека пользователь может легко отменить ошибочные действия. В настоящее время стандартным подходом к реализации данной функции являются два паттерна проектирования, Memento («Хранитель») и Command («Команда»), с различной трудоемкостью реализации и эффективностью использования ресурсов компьютера [1, 2].

Паттерн Memento основан на пошаговом запоминании состояний редактируемой модели после каждого выполненного пользователем дейст-

вия. Его реализация обычно достаточно проста и имплементируется посредством полной сериализации модели в бинарный архив, который затем помещается в стек состояний. Другим преимуществом данного паттерна является простота восстановления любого сохраненного состояния модели в любой момент времени (через десериализацию соответствующего архива).

На рис. 1 (см. вклейку) представлены схемы работы паттерна Memento. При каждом изменении модели данных происходит помещение нового состояния в стек отмены (рис. 1, *а*). При этом указатель стека позиционируется на актуальное в данный момент состояние модели. При выполнении пользователем команды отмены (Undo) происходит смещение указателя стека на одну позицию «вниз» и восстановление состояния модели по данному индексу (рис. 1, *б*). Соответственно при выполнении команды повтора (Redo), указатель стека смещается на одну позицию «вверх». Если в какой-либо момент времени пользователь выполнит редактирование модели, то все состояния стека отмены, находящиеся «выше» текущего указателя, будут перезаписаны текущим состоянием модели данных.

Пример формирования стека с применением паттерна Memento приведен в табл. 1. Для наглядности в качестве исходных данных взят произвольный короткий текст на английском языке. Как видно из табл. 1, главным недостатком паттерна Memento является большой объем потребляемой памяти, так как каждое состояние модели должно быть сохранено заново, независимо от того, насколько значительно оно изменилось в результате действий пользователя. Для того чтобы уменьшить расход памяти, часто используется сжатие сериализированных архивов и их распаковка при десериализации, но эффективность данного способа существенно зависит от используемого алгоритма сжатия, объема и структуры данных. Поэтому паттерн Memento целесообразно использовать в системах, в которых каждая операция над моделью приводит к значительному

Таблица 1

Шаг	Модель данных	Выполненная пользователем операция	Состояние стека
1	It is a text which is to be edited	Исходные данные до редактирования	It is a text which is to be edited
2	It is a text to be edited	Удалена часть данных из позиций от 12 по 20 (9 символов)	It is a text to be edited
3	It is some text to be edited	Замена одного символа «a» в позиции 6 на четыре символа «some»	It is some text to be edited
4	It is some longer text to be edited	Добавлена последовательность «longer» в позиции 11	It is some longer text to be edited

ее изменению (например, изменение графического изображения посредством фильтров). Если изменения модели незначительны относительно неизменяемого объема данных, либо изменения носят строго дефинированный характер, то следует рассматривать возможность применения других методов.

Паттерн Command основан на сохранении списка модификаций (команд), применяемых к исходной модели. Каждое действие пользователя над моделью описывается в виде команды, которая применяется к предыдущему состоянию модели, и ее параметров, необходимых для выполнения модификации. В стек изменений, в отличие от паттерна Memento, заносятся только коды команд и наборы параметров, что, как правило, занимает относительно небольшой объем памяти. Классический пример использования паттерна Command — графические редакторы (типа Adobe Photoshop), текстовые процессоры (Microsoft Word, OpenOffice) и другие программы, в которых изменения модели данных выполняются соответственно строго регламентированным действиям пользователя.

Рассмотрим пример формирования стека с применением паттерна Command (табл. 2). В стек на каждом шаге заносится код одного из возможных действий над исходной моделью данных (использовано три команды: удаление — DELETE, замена — REPLACE, вставка — INSERT). Каждому действию соответствует определенный набор параметров. Так, в примере каждая из команд первым параметром определяет начальную позицию в исходном тексте, вторым параметром для команды удаления и замены указана длина исходной последовательности, последним параметром для команд вставки и замены являются соответственно вставленный либо измененный пользователем текст.

Очевидно, восстановление предыдущих состояний требует последовательного применения всех сохраненных команд к исходному (базовому) состоянию модели, пока не будет достигнута нужная трансформация.

Таблица 2

Шаг	Модель данных	Выполненная пользователем операция	Состояние стека
1	It is a text which is to be edited	Исходные данные до редактирования	—
2	It is a text to be edited	Удалена часть данных из позиций от 12 по 20 (9 символов)	DELETE, 12, 9
3	It is some text to be edited	Замена одного символа «a» в позиции 6 на четыре символа «some»	REPLACE, 6, 1, «some»
4	It is some longer text to be edited	Добавлена последовательность «longer» в позиции 11	INSERT, 11, «longer»

Также очевидно, что в зависимости от имплементации команд и размера стека для восстановления нужного состояния модели может потребоваться значительное число операций. Так, если пользователь, находясь в шаге 4, решит отменить вставку слова «longer», то алгоритм должен будет последовательно выполнить команды от 1 до 3, что неэффективно. Поэтому на практике применяется модификация метода, в которой для запоминания изменений между состояниями модели используются два стека: стек отмены и стек повтора.

При выполнении пользователем того или иного действия в стек отмены заносится код команды и параметры для трансформации текущего состояния модели в состояние до изменения, а в стек повтора — команда обратной трансформации из предыдущего в текущее состояние (рис. 2, см. вклейку). При этом указатель стека отмены перемещается на позицию, соответствующую вершине стека, а указатель стека повтора находится на одну позицию выше вершины стека (команды повтора неактивны, пока пользователь не выполнил действие отмены). В случае выполнения пользователем операции отмены к текущему состоянию модели данных применяется соответствующая «обратная» команда, а в список повторов будет занесена команда, обратная ей. Соответственно при выполнении пользователем операции повтора обратные команды применяются к модели и удаляются из данного списка.

Основной недостаток данного метода — значительная (по сравнению с Memento) трудоемкость реализации, так как, во-первых, каждое действие пользователя необходимо описать соответствующей командой-модификатором, и, во-вторых, для каждой команды должна быть создана обратная ей операция, что не всегда возможно. Например, если команда представляет собой необратимое действие наподобие графического фильтра, то обратная команда, по сути, должна полностью содержать предыдущее состояние модели.

Каждая операция редактирования, выполненная пользователем, приводит к изменению состояния модели данных. Если обозначить состояние модели до изменений через M_i, а состояние модели после редактирования — через M_{i+1}, то любое действие пользователя по изменению модели можно обозначить как функцию трансформации: $M_i \rightarrow M_{i+1}$, или $M_{i+1} = F(M_i)$. Соответственно для восстановления предыдущего состояния модели из M_{i+1} в M_i (при выполнении операции отмены) необходимо выполнить обратную трансформацию $M_{i+1} \rightarrow M_i$, что может быть выражено применением к текущему состоянию модели обратной функции трансформации: $M_i = F^{-1}(M_{i+1})$.

Как указано выше, для паттерна Command трансформации модели реализуются в виде отдельных команд-модификаторов, каждая из которых

описывает конкретную трансформацию $M_i \to M_{i+1}$, и таким же образом — обратную команду для трансформации $M_{i+1} \to M_i$. При достаточной эффективности данного паттерна в действии существенной проблемой является значительная трудоемкость имплементации, так как каждая команда, как правило, реализуется отдельным классом, описывающим конкретные трансформации модели. Данная особенность паттерна Command не позволяет реализовать универсальный алгоритм для управления стеком недавних действий, который можно было бы применять для различных моделей данных без существенной модификации кода.

Идея предлагаемого метода Diff заключается в автоматическом генерировании команд-модификаторов на основе различий между предыдущим и текущим состояниями модели данных. Если рассматривать каждое состояние модели данных как некую абстрактную последовательность элементов (например, сериализированную в бинарный архив, как это предполагается в паттерне Memento), то можно выделить семь основных вариантов модификаций абстрактного представления модели данных, которые выполняются на каждом шаге редактирования ее пользователем (рис. 3, см. вклейку).

1. Однократное изменение. Предполагается, что в ходе редактирования модели был изменен одиночный параметр фиксированного размера. Соответственно сериализированные состояния модели различаются только на один (либо несколько) байт, расположены в архиве последовательно и имеют одинаковый начальный индекс в обоих архивах, имеющих одинаковую длину.

2. Однократная вставка. Как правило, это результат операции по расширению модели данных (добавление новых объектов либо параметров). В этом случае текущее состояние архива полностью содержит в себе предыдущее состояние, дополненное некоторой последовательностью байт, начиная с произвольного индекса.

3. Однократное удаление. Обычно это является результатом операции редукции модели данных (удаление объектов либо параметров). Фактически это операция, обратная однократной вставке. Текущее состояние архива повторяет предыдущее состояние, за исключением отсутствия некоторой последовательности байт.

4. Однократная замена. Часто это является результатом переименования какого-либо параметра либо редактирования текстового значения. Данная модификация близка к однократному изменению с той лишь разницей, что оба архива отличаются один от другого наличием уникальных последовательностей различной длины, а также индексами начала.

5. Однократное перемещение. Результат операции обмена последовательности данных в пределах модели (перемещение объекта, изменение индексов и др.). При этом оба архива сохраняют свою длину, но две произвольные последовательности данных (не обязательно размещенные последовательно относительно одна другой) изменяют свое взаимное расположение.

6. Многократная модификация. Результат операции, примененной одновременно ко множеству объектов либо параметров модели (полнотекстовый поиск и замена, перемещение группы объектов, многократное переименование и др.). В результате оба архива содержат в себе множественные изменения относительно друг друга, которые не подлежат универсальному определению, однако, в то же время, в них можно выделить общие последовательности.

7. Глобальная модификация. Является «worst case» (т.е. худшим случаем применительно к рассматриваемому методу), поскольку операция над моделью данных предполагает практически полное изменение модели (например, графический фильтр для всего изображения, создание новой модели либо загрузка данных из файла). В этом случае изменения между архивами настолько значительны, что не существует возможности выделить какие-либо общие последовательности данных в них.

Модификации *1—4*, по сути, отражают операции редактирования модели, выполняемые последовательно. Поскольку при этом на каждом шаге редактирования модель изменяется пользователем, как правило, незначительно, различия между состояниями модели также будут минимальными по отношению к общему объему данных (например, при изменении только одного параметра модели). Как видно из рис. 3, *1—4*, каждое состояние модели до и после редактирования фактически разбивается модифицированной последовательностью на две части (слева и справа), которые остаются неизменными. В этих случаях достаточно эффективным будет применение двунаправленного сравнения обоих состояний модели до первого несовпадения. В зависимости от позиции найденного несовпадения (или его отсутствия) можно сделать вывод о том, какие модификаторы должны быть применены к последовательностям.

Принимая во внимание гомогенность модификаций *1—4*, их можно описать одной из возможных операций над двоичным массивом [3]:

добавление (либо вставка) — исходная последовательность байт дополняется другой последовательностью произвольной длины в произвольной позиции;

удаление — из исходной последовательности удаляется последовательность байт произвольной длины, начиная с произвольной позиции;

замена — одна произвольная последовательность в исходном массиве заменяется другой.

Таким образом, для формирования универсальных модификаторов для вариантов *1—4* достаточно трех команд, каждая из которых обладает фиксированным набором параметров: начальный индекс в исходном массиве, длина последовательности (при удалении или замене) и собственно измененные данные (при вставке или замене). Каждой прямой трансформации соответствует аналогичная обратная трансформация, которая преобразует результирующий массив байт обратно в исходный (каждая вставка обратна удалению и наоборот, а каждая замена — обратной замене).

Пример формирования стека по данному алгоритму приведен в табл. 3.

Очевидно, что для вариантов *5—7* алгоритм двунаправленного поиска окажется в общем случае неэффективным, так как он в состоянии определить только начало и конец изменений в пределах целого архива. Вариант *7* исключает генерацию какого-либо универсального модификатора. При этом могут быть рассмотрены следующие возможные решения:

1. Помещение в стек модификаторов, описывающих конкретную трансформацию отдельными командами по аналогии с паттерном Command. Это решение не является универсальным, однако оно может быть достаточно эффективным относительно расхода памяти.

2. Применение алгоритмов компрессии данных и сохранение в стеке двух состояний модели в сжатом виде (аналогично паттерну Memento). Такое решение обеспечивает универсальность алгоритма в результате эффективного использования памяти и быстродействия.

Вариант *6* может быть распознан с помощью алгоритмов поиска одинаковых последовательностей в массиве (например, алгоритма поиска наиболее длинной общей последовательности LCS (Longest Common Se-

Таблица 3

Шаг	Модель данных	Выполненная операция	Команда повтора	Список команд отмены
1	It is a text which is to be edited	Исходные данные до редактирования	—	—
2	It is a text to be edited	Удалена часть данных из позиций от 12 по 20 (9 символов)	DELETE, 12, 9	INSERT, 12, «which is »
3	It is some text to be edited	Замена одного символа «a» в позиции 6 на четыре символа «some»	REPLACE, 6, 1, «some»	REPLACE, 6, 4,«a»
4	It is some longer text to be edited	Добавлена последовательность «longer» в позиции 11	INSERT, 11, «longer»	DELETE, 11, 6

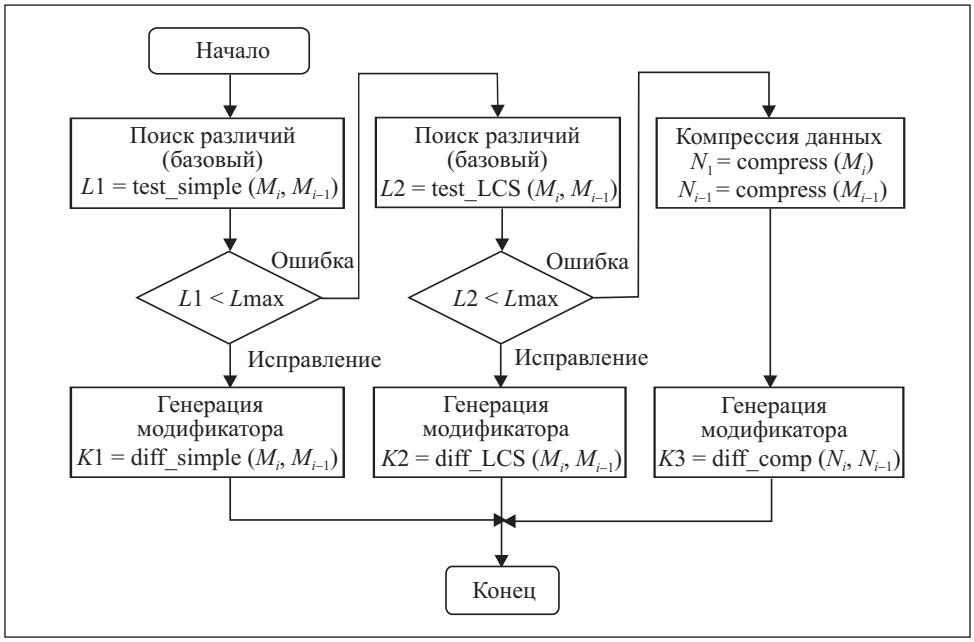

Рис. 4. Обобщенный алгоритм генерации модификаторов: M_{i-1} и M_i — исходное и текущее состояния модели данных (до редактирования и после); L_{max} — максимально допустимая длина модификатора (в байтах), взятая как критерий оптимальности результата работы алгоритма поиска; test_simple — определение размера модификатора, получаемого после выполнения простого поиска различий; diff_simple — генерирование модификатора с помощью простого поиска различий; test_LCS и diff_LCS — определение размеров и генерирование модификаторов как результата расширенного поиска различий; compress — сжатие данных; diff_comp — генерирование модификаторов, помещающих сжатые данные в стек отмены

quence) [4]). Если данный поиск дал лучшие результаты (были выявлены несколько незначительных различий, составляющих небольшой процент от общего объема данных), то в стек будет помещен модификатор множественной замены (REPLACE_MULTI), параметры которого описывают каждое из необходимых изменений, последовательно применяемых к модели.

Вариант *5* также требует дополнительных вычислительных затрат, чтобы быть распознанным в двоичном архиве. Возможным решением в этом случае является использование алгоритма поиска перестановок (SoP, Search of Permutations), например алгоритма Хипа [5]. В случае позитивного результата в стек будет помещен модификатор перемещения (MOVE), содержащий в качестве параметров индексы начала и длины обеих перемещенных последовательностей.

Следует заметить, что с использованием алгоритмов LCS и SoP может быть затрачено значительное время на обработку больших объемов дан-

ных, что приведет к уменьшению времени отклика системы на реакцию пользователя, а следовательно, вызовет негативное мнение с его стороны об эффективности визуального редактора модели в целом. Поэтому, в зависимости от объемов редактируемых данных, в случае негативного результата алгоритма быстрого поиска изменений (для вариантов *1—4*) целесообразно не расходовать ресурсы на дополнительную оценку последовательностей, а применить алгоритм сжатия общего назначения аналогично паттерну Memento (для вариантов *5* и *6* сжатие данных может быть эффективным в случае, если отношение суммарной длины измененных последовательностей к объему неизмененных — достаточно мало).

На рис. 4 представлен обобщенный алгоритм Diff. Наиболее эффективным представляется использование метода Diff для реализации стека недавних состояний в системах, которые незначительно модифицируют модель данных при каждом отдельном шаге редактирования (например, редактирование текстовых данных либо численных параметров). По сравнению с паттерном Memento метод Diff обеспечивает значительное сокращение расхода памяти в большинстве случаев (кроме вариантов *5—7*, когда в зависимости от исходных данных и дополнительных алгоритмов поиска конечный расход памяти может быть приблизительно одинаковым). По сравнению с паттерном Command метод Diff обеспечивает полную универсальность реализации (единый алгоритм без необходимости добавления новых команд при расширении функциональности редактора модели).

Критерии, определяющие эффективность методов реализации стека недавних действий, следующие:

относительно работы метода — L — объем памяти, необходимый для хранения данных в стеке; T_{ux} — время, затраченное на выполнение x шагов отмены; T_{rx} — время, затраченное на выполнение x шагов повтора; T_m — время, затраченное на формирование и помещение в стек данных одного шага изменения модели;

относительно имплементации метода — C — относительная сложность имплементации метода; T_c — время, затраченное разработчиком на имплементацию; U — универсальность имплементации метода (возможность его повторного использования в других приложениях).

Таким образом, общий критерий эффективности работы (производительности) метода может быть определен так:

$$E_p = \begin{cases} L \to \min,\ L < RAM_{\max}, \\ T_{ux} \to \min,\ T_{ux} < T_{x\max}, \\ T_{rx} \to \min,\ T_{rx} < T_{x\max}, \\ T_m \to \min,\ T_{ux} < T_{s\max}, \end{cases}$$

где RAM_{max} — максимально допустимое количество памяти, выделяемое на стек недавних действий; $T_{x\,max}$ — максимально допустимое время ожидания пользователем завершения x операций восстановления данных из стека; $T_{s\,max}$ — максимально допустимая временная пауза, возникающая после редактирования модели (в том числе время, затраченное на помещение текущего состояния модели в стек).

Критерий эффективности имплементации запишем в виде

$$E_i = \begin{cases} C \to \min, \\ T_c \to \min, \\ U \to \max. \end{cases}$$

Значения $T_{x\,max}$ и $T_{s\,max}$ — субъективные ограничения, зависящие от многих факторов: объема данных, применяемых алгоритмов, быстродействия целевой системы, глубины стека и др. Основополагающим показателем эффективности в этом случае является восприятие пользователем возникающих задержек при выполнении им тех или иных действий. Если пользователь отменяет одновременно несколько достаточно трудоемких операций, то приемлемое для него время $T_{x\,max}$ при $x > 1$ может быть значительно большим, чем $T_{x\,max}$ при $x = 1$ (т.е. при отмене либо повторе последнего действия). Как правило, пользователь ожидает мгновенной реакции на отмену либо повтор одного действия, так же как и на изменение модели ($T_{s\,max}$). Возникающие при этом заметные задержки (в среднем приемлемым временем реакции диалогового средства на действия пользователя принято считать время до 250 мс) будут раздражать пользователя и вызывать у него негативное отношение к программному продукту, что является недопустимым отрицательным фактором.

Ограничение по памяти RAM_{max} — важный параметр, от которого зависит максимальная глубина стека последних операций. Чем эффективнее используется память имплементируемым алгоритмом, тем меньший объем памяти займут данные, сохраняемые в стеке после каждого редактирования модели, и соответственно тем больше действий сможет отменить пользователь.

Универсальность имплементации U определяет возможность повторного использования уже написанного кода для применения метода в других системах без изменений (либо с минимальной адаптацией).

Критерий эффективности имплементации E_i непосредственно влияет на временную и экономическую составляющие разрабатываемого проекта. Чем меньше времени затратят разработчики на написание и возможное

Таблица 4

Критерий	Паттерн Memento	Паттерн Command	Метод Diff
L	Объем памяти L_m в общем случае больше, чем требуют другие методы: $L_m > L_c$; $L_m > L_d$	Объем памяти L_c соответствует параметрам одной команды и в общем случае минимален: $L_c \to \min$	Объем памяти L_d зависит от состояния модели до и после редактирования, но в общем случае незначительно превышает L_c: $L_m > L_d > L_c$; $L_d \to L_c$
T_{ux}	$T_{ux.m}$ равно времени однократного восстановления состояния модели из архива	$T_{ux.c}$ равно времени выполнения x команд отмены	$T_{ux.d}$ равно времени выполнения x команд отмены
T_{rx}	Аналогично $T_{ux.m}$	Аналогично $T_{ux.c}$ для x команд повтора	Аналогично $T_{ux.d}$ для x команд повтора
T_m	$T_{m.m}$ равно времени сериализации параметров модели T_s в архив (и возможного сжатия T_c): $T_{m.m} = T_s + T_c$	$T_{m.c}$ равно времени сериализации параметров команды T_{sc} и в общем случае минимально: $T_{m.c} = T_{sc} \to \min$	$T_{m.d}$ равно времени сериализации параметров модели T_s в архив, поиска различий между состояниями моделей T_f и сериализации параметров сгенерированной команды T_{sgc}: $T_{m.d} = T_s + T_f + T_{sgc}$
C	$C_m \to \min$; $C_m = \text{const}$ (универсальная и наиболее простая имплементация в общем случае)	$C_c \to \max$ (наиболее трудоемкая имплементация, зависит от числа возможных команд N_c)	Имплементация универсальна, трудоемкость незначительна $C_d \to C_m$; $C_d < C_c$; $C_d = \text{const}$
T_c	$T_{c.m} = \text{const}$; $T_{c.m} \to \min$	$T_{c.c} = T^* N_c$; $T_{c.c} \to \max$	$T_{c.d} = \text{const}$; $T_{c.d} \to T_{c.m}$
U	$U_m \to \max$	$U_c \to \min$ в общем случае (существенно зависит от типа приложения)	$U_d \to \max$

Таблица 5

Критерий	Memento	Command	Diff
L	1	3	2
T_{ux}, T_{rx}	3	2	2
T_m	2	3	2
C, T_c	3	1	3
U	3	2	3

расширение программного кода, реализующего стек последних действий, тем больше будет экономия средств, выделенных на разработку проекта, что означает уменьшение его стоимости для конечных пользователей.

Рассмотренные критерии эффективности для различных методов описаны в табл. 4, из которой следует, что каждый указанный метод имеет свои достоинства и недостатки. Оценив указанные критерии по трехбалльной шкале (1 — худший результат, 2 — средний, 3 — лучший), можно получить условную характеристику рассмотренных методов, которая представлена в табл. 5.

Выводы

Характеристики эффективности по быстродействию и расходу памяти предложенного метода Diff по сравнению с методами Memento и Command, в общем случае, требуют улучшения. Однако метод Diff не требует значительных затрат на разработку, так как алгоритм формирования модификаторов команд индифферентен к сложности модели и действиям по ее изменению. Практическая имплементация метода Diff доказывает его универсальность и эффективность при разработке визуальных средств редактирования моделей данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Гамма Э., Хелм Р., Джонсон Р., Влиссидс Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. — Addison-Wesley: «Питер», 1994. — 395 с.
2. *Паттерны* проектирования (Design Patterns). [Электронный ресурс]. — Режим доступа: http://www.cpp-reference.ru
3. *Масюк А.Л.* Реализация стека недавних действий для визуальных средств редактирования моделей данных / Сб. тр. Пятой международной конференции «Моделирование-2016». — Киев: ИПМЭ им. Г.Е. Пухова НАН Украины, 2016. — 292 с. — С. 101—104 (ISBN 978-966-02-7928-5).
4. *Chvatal Václáv, Sankoff David* Longest common subsequences of two random sequences// Journal of Applied Probability. — 1975. — № 12. — P. 306—315.
5. *Heap B.R.* Permutations by Interchanges (PDF)// The Computer Journal. — 1963. — **6** (3): 293–4. doi:10.1093/comjnl/6.3.293

REFERENCES

1. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994), *Priyomy obektno-orientirovannogo proektirovaniya. Patterny proektirovaniya* [Design patterns. Elements of reusable object-oriented software], Addison-Wesley, Piter.
2. *Patterny proektirovaniya* [Design patterns], available at: http://www.cpp-reference.ru.
3. Masyuk, A.L. (2016), “Implementation of undo-redo stack of visual data model editors”, *Trudy 5-oi Mezhdunarodnoi konferentsii Modelorovanie-2016* [Proceedings of the 5th International Scientific Conference Simulation-2016], Pukhov Institute for Problems of Modeling in Energy Engineering, NAS of Ukraine, Kiev, pp. 101-104, ISBN 978-9660279285.

4. Chvatal, V. and Sankoff, D. (1975), “Longest common subsequences of two random sequences”, *Journal of Applied Probability*, Vol. 12, pp. 306-315.
5. Heap, B.R. (1963), “Permutations by interchanges” (PDF), *The Computer Journal*, Vol. 6, no. 3, pp. 293-294, doi:10.1093/comjnl/6.3.293.

A.L. Masyuk

DIFF METHOD FOR IMPLEMENTING UNDO STACK
OF THE RECENT USER ACTIONS

This article deals with the most common patterns Memento and Command which are used nowadays in order to implement a recent user actions undo stack. The new method Diff is suggested, which combines advantages of the both patterns and allows quite simple and common implementation of the undo stack for a developer. The method has been practically tested while implementing a visual editor of the mine ventilation model graphs (as well as some other applications). Usage of the applications has proved the efficiency of the Diff method by such criteria as speed of response, memory consumption, code reuse and difficulty of the final implementation.

K e y w o r d s: visual editing, undo stack, graphical interface.

Поступила 30.08.16;
после доработки 14.11.16

МАСЮК Арсений Леонидович, аспирант кафедры компьютерной инженерии факультета компьютерных наук и технологий Донецкого национального технического университета, который окончил в 2002 г. Область научных исследований – параллельные вычислительные системы, интерактивные диалоговые алгоритмы.

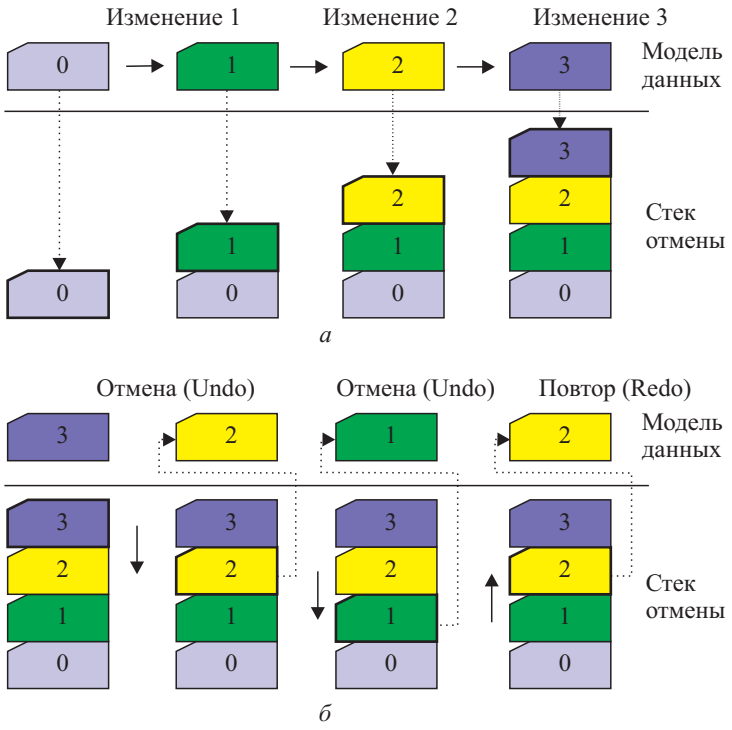

Рис. 1. Паттерн Memento при изменении модели данных (*а*) и при выполнении команд отмены и повтора (*б*)

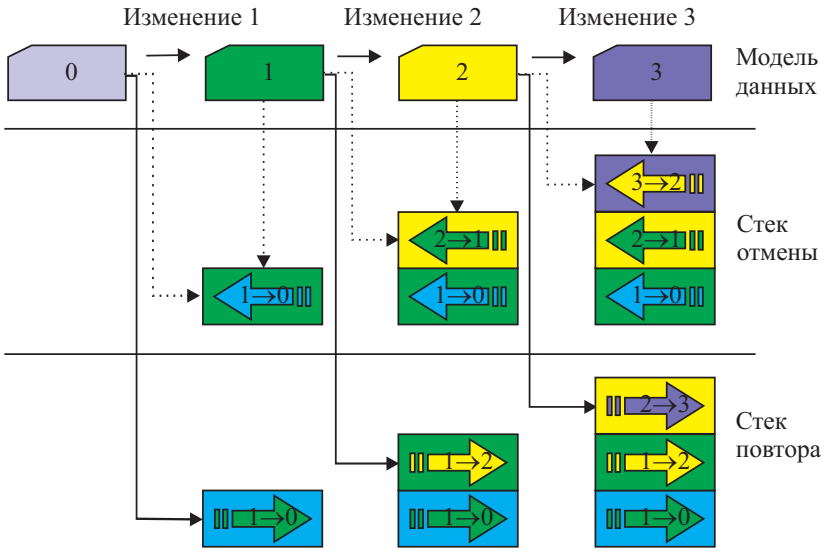

Рис. 2. Формирование стеков отмены и повтора паттерном Command

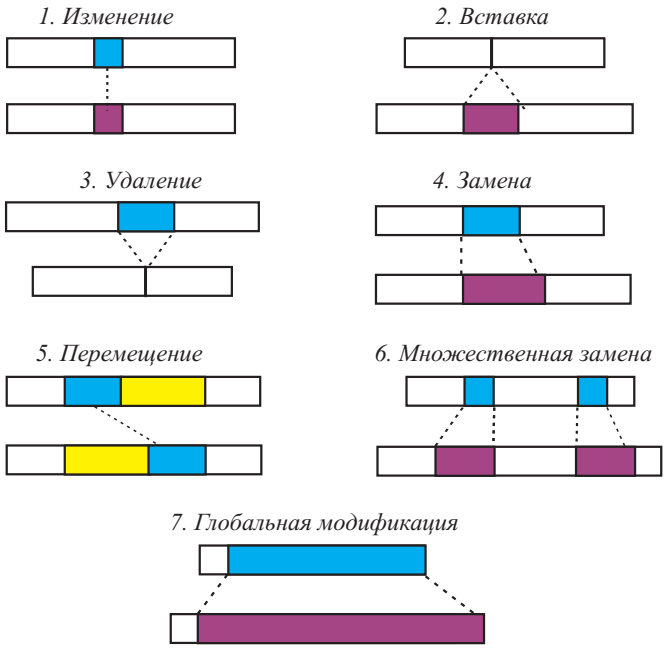

Рис. 3. Возможные модификации абстрактной модели данных: верхняя часть каждого рисунка соответствует модели данных до изменения, нижняя — после изменения