Статья опубликована в журнале «Компьютерные инструменты в образовании». 2005. № 3, с. 62 – 76.

Казаков Матвей Алексеевич
Шалыто Анатолий Абрамович

АВТОМАТНЫЙ ПОДХОД К РЕАЛИЗАЦИИ АНИМАЦИИ В ВИЗУАЛИЗАТОРАХ АЛГОРИТМОВ

ВВЕДЕНИЕ

Визуализаторы алгоритмов широко используются в процессе преподавания дискретной математики и теории классических вычислительных алгоритмов [1 – 3].

Визуализатор — программа, иллюстрирующая выполнение алгоритма при определенных исходных данных [4].

В работе [5] рассматривался метод построения визуализаторов на основе автомата Мили, а также предлагалась технология формального преобразования вычислительного алгоритма в визуализатор. Аналогичным образом логика визуализаторов может строиться на основе автомата Мура.

Ранее визуализатор рассматривался, как дискретная последовательность статических иллюстраций, что в большинстве случаев достаточно для обучения. Однако, в некоторых алгоритмах статических иллюстраций мало. Примером такого алгоритма может служить пирамидальная сортировка (сортировка с помощью кучи) [6], так как процессы передвижения значений между вершинами дерева, в виде которого представлен массив, наиболее наглядно демонстрируется при помощи анимации.

В настоящей работе предлагается расширение технологии построения визуализаторов с целью включения в нее возможности анимации требуемых шагов визуализации. При этом рассматривается такая разновидность анимации, при которой изображается переход от предыдущих значений визуализируемых переменных к последующим. Поскольку каждая статическая иллюстрация является функцией от визуализируемых переменных, то указанная анимация и будет визуализировать соответствующий шаг алгоритма в динамике. Предлагаемый подход иллюстрируется на примере традиционной реализации алгоритма пирамидальной сортировки.

1. АВТОМАТНЫЙ ПОДХОД К ПОСТРОЕНИЮ ВИЗУАЛИЗАТОРОВ БЕЗ АНИМАЦИИ

В работе [5] описана автоматная технология построения визуализаторов алгоритмов, состоящая из следующих этапов.

1. **Постановка задачи, которую решает визуализируемый алгоритм.**
2. **Решение задачи в словесно-математической форме**.
3. **Выбор визуализируемых переменных**.
4. **Анализ алгоритма для визуализации.** Анализируется решение с целью определения того, что и как отображать на экране.

5. **Алгоритм решения задачи**.
6. **Реализация алгоритма на выбранном языке программирования.** На этом шаге производится реализация алгоритма, его отладка и проверка работоспособности.
7. **Построение схемы алгоритма по программе**.
8. **Преобразование схемы алгоритма в граф переходов автомата, реализующего алгоритм**, который может быть как автоматом Мили, так и автоматом Мура [8, 9].
9. **Преобразование автомата реализующего алгоритм в автомат визуализации**.
10. **Выбор интерфейса визуализатора.**
11. **Сопоставление иллюстраций и комментариев с состояниями автомата** (иллюстрации формируются компонентом программы, называемым «рисовальщик»).
12. **Архитектура программы визуализатора**.
13. **Программная реализация визуализатора.**

С целью обеспечения возможности анимации в последовательность действий вводятся дополнительные шаги. Отметим, что поскольку изменения визуализируемых переменных в используемых в настоящей работе автоматах Мура производятся в состояниях (что весьма удобно), то будем рассматривать вопрос о введении анимации в визуализаторы, построенные на основе автоматов Мура. В виду того, что при введении анимации в автомате визуализации появляются действия на дугах, то этот автомат становится смешанным.

2. ДОПОЛНИТЕЛЬНЫЕ ЭТАПЫ ДЛЯ ОБЕСПЕЧЕНИЯ АНИМАЦИИ В ВИЗУАЛИЗАТОРАХ

Для построения визуализатора с анимацией предлагается применять четыре типа автоматов:

- **автомат, реализующий алгоритм** (формируется на восьмом этапе, указанном выше);
- **автомат визуализации** (формируется на девятом этапе, указанном выше);
- **преобразованный автомат визуализации**;
- **автомат анимации**.

Автомат визуализации позволяет отображать **статические иллюстрации** в каждом состоянии, что приводит к **статической (пошаговой) визуализации**. Под статической иллюстрацией понимается фиксированный кадр, не изменяющийся с течением времени.

Преобразованный автомат визуализации кроме статических иллюстраций отображает также и **динамические иллюстрации** в дополнительно вводимых анимационных состояниях. Это позволяет говорить о **динамической визуализации (анимации)**.

Динамическая иллюстрация состоит из набора промежуточных статических иллюстраций, отображаемых на экране **автоматом анимации** через определенные промежутки времени. Это приводит к динамическому изменению иллюстрации в анимационном состоянии.

2.1. Дополнительные этапы технологии для обеспечения анимации

Для автоматов Мура анимацию естественно проводить (также как и формирование статических изображений и комментариев) в состояниях — несмотря на то, что в визуализаторе анимируемые процессы «текут», в автомате каждый из них реализуется только в одном состоянии. Для введения анимации в визуализатор расширим технологию за счет следующих шагов:

a) **выбор состояний автомата визуализации, в которых выполняется анимация** (такие состояния будем назвать анимационными);
b) **построение преобразованного автомата визуализации путем замены каждого из анимационных состояний тремя состояниями**, в первом из которых производятся преобразования данных, во втором выполняется анимация, соответствующая анимационному состоянию, а в третьем — отображается статическая иллюстрация;
c) **разработка анимационного автомата** (выполняется один раз для всех визуализаторов, проектируемых по этой технологии);
d) **обеспечение взаимодействия между преобразованным автоматом визуализации и анимационным автоматом;**
e) **разработка и реализация функции вывода каждой динамической иллюстрации, зависящей от состояния автомата визуализации и шага анимации.**

2.2. Формализация построения автомата визуализации

До перехода к более подробному рассмотрению новых этапов изложим, в чем состоит девятый этап технологии (рис. 1).

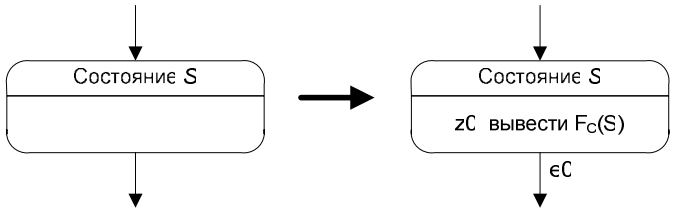

Рисунок 1. *Формирование состояний автомата визуализатора*

На этом этапе в каждое визуализируемое состояние вводится выходное воздействие *z0*, осуществляющее формирование статического изображения и комментария в рассматриваемом состоянии. Различия в изображениях и комментариях формализуются с помощью функции $F_C(S)$, описанной ниже, параметром которой является номер состояния *S*, а обозначение *C* указывает, что это статическая функция. В каждом состоянии, формирующем выходное воздействие *z0*, переход к следующему состоянию осуществляется по событию *e0* (нажатие клавиши «шаг» в интерфейсе визуализатора).

2.3. Введение анимационных состояний в автомат визуализации

Поясним каждый из вновь вводимых этапов *a* – *e*.

На этапе *a* выполняется выбор анимационных состояний в соответствии с особенностями алгоритма и принятыми решениями по анимации на четвертом этапе исходной технологии («Анализ алгоритма для визуализации»). Так как в алгоритме пирамидальной сортировки, который используется ниже в качестве примера, наибольший интерес представляет процесс движения элементов по вершинам «пирамиды», то состояния, в которых происходит переход, и будут выбраны в качестве анимационных.

На этапе *b* в автомате визуализации для получения преобразованного автомата визуализации каждое анимационное состояние формально заменяется тремя состояниями (рис. 2).

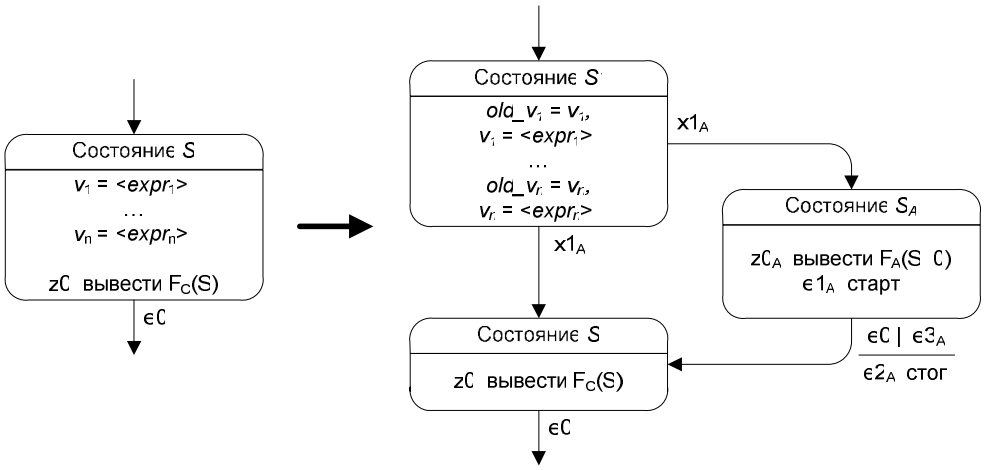

Рисунок 2. *Преобразование анимационного состояния*

Опишем это преобразование. Предположим, что состояние S выбрано для визуализации. Предположим также, что в этом состоянии изменяются значения визуализируемых переменных $v_1, \dots, v_n$. При этом переменной v_i присваивается значение выражения $< expr_i >$. Преобразование состоит в замене выбранного состояния на три состояния — S' (операционное состояние), S_A (анимационное состояние) и S (завершающее состояние).

В состоянии S' значения переменных $v_1, \dots, v_n$ предварительно сохраняются в переменных $old\_v_1, \dots, old\_v_n$, а также проводятся все действия, выполняемые в состоянии S автомата визуализации.

В состоянии S' преобразованного автомата визуализации в отличие от состояния S исходного автомата этого типа не происходит ожидания события, а выполняется переход в одно из состояний S_A или S.

Переход в состояние S_A производится, если переменная $x1_A$ принимает значение `true` (анимация включена). Анимация осуществляется, пока преобразованный автомат визуализации находится в состоянии S_A. При входе в это состояние осуществляется запуск автомата анимации при помощи события $e1_A$: *старт* и отображается иллюстрация $F_A(S, 0)$, соответствующая началу анимации.

Выход из состояния анимации и переход в состояние S происходит, как по событию $e0$ (нажатие клавиши), так и по событию $e3_A$ (окончание анимации), и сопровождается сигналом $e2_A$: *стоп*. Таким образом, обеспечивается как автоматическое, так и ручное завершение анимации.

Как отмечено выше, состояние S является завершающим. В нем выполняется отображение статической иллюстрации $F_C(S)$. Отметим, что поскольку в преобразованном анимационном автомате присутствуют действия на дугах, то этот автомат является смешанным автоматом.

Следует отметить, что при выключении анимации (переменная $x1_A$ принимает значение `false`) преобразованный автомат визуализации становится изоморфным исходному автомату визуализации. Поэтому включение анимации может рассматриваться именно как ***расширение*** исходного визуализатора, а не его преобразование.

2.4. Построение автомата анимации

На этапе *c* строится автомат анимации (рис. 3). Следует отметить, что он является смешанным автоматом, поскольку содержит действия, как на дугах, так и в вершинах графа перехода.

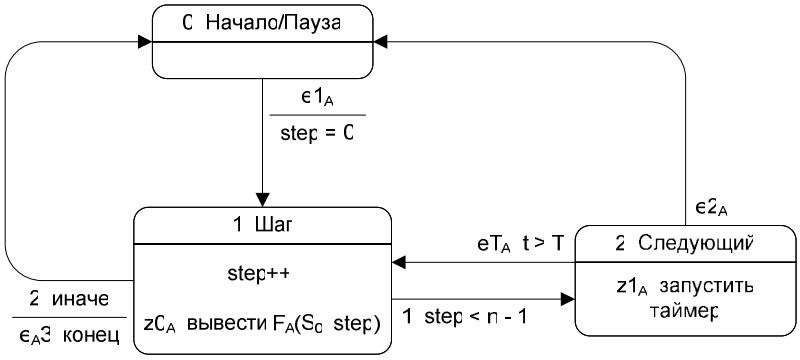

Рисунок 3. *Автомат анимации*

Из рассмотрения этого рисунка следует, что автомат предназначен для изменения значения переменной *step*, которая хранит номер шага анимации.

Автомат анимации формирует выходные воздействия и событие:

- $z1_A$*: запустить таймер* — запускает таймер, который через период времени *T* генерирует событие eT_A;
- $z0_A$*: вывести* $F_A(S_0, step)$ — информирует «рисовальщик» о необходимости перерисовать иллюстрацию;
- $e3_A$*: конец анимации* — сигнализирует о том, что анимация закончилась.

Анимационный автомат реагирует на следующие события:

- $e1_A$: *старт* — автомат визуализации перешел в состояние, в котором должна начаться анимация;
- $e2_A$: *стоп* — автомат визуализации перешел в состояние, в котором анимация должна закончиться;
- eT_A — событие от таймера, по которому осуществляется переход к следующему шагу анимации.

Описанный автомат является универсальным и может быть использован в любом визуализаторе.

На этапе *d* автомат визуализации и автомат анимации связываются посредством событий и выходных воздействий, как показано на рис. 4.

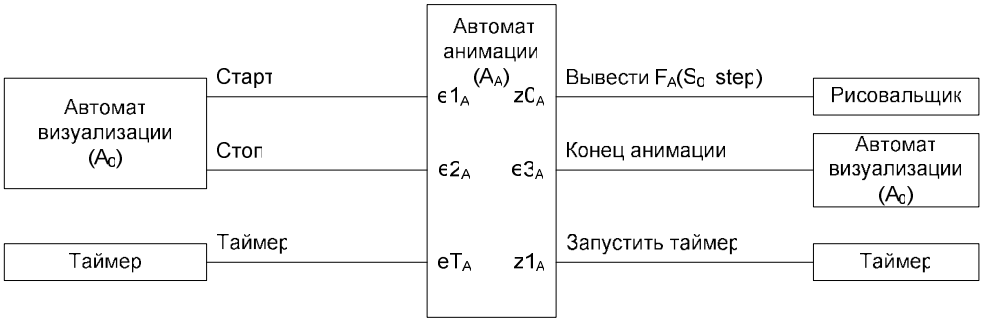

Рисунок 4. *Схема связей автомата анимации*

2.5. Обеспечение отображения анимации

На этапе *e* стандартная функциональность «рисовальщика» сопоставления номера состояния с текстовым комментарием и статической иллюстрацией расширяется. В режиме анимации «рисовальщик» принимает на вход значение номера шага. Другими словами, если функция F_C реализует статические иллюстрации, а функция F_A — динамические (анимационные) иллюстрации, то при переходе от статики к анимации осуществляется преобразование вида:

$$F(state) \to \begin{cases} F_C(state); \\ F_A(state, step). \end{cases}$$

При этом справедливы следующие соотношения:

$$F_C(state) = F(state); \quad (1)$$

$$F_A(state, 0) = F_C(state - 1); \quad (2)$$

$$F_A(state, N) = F_C(state). \quad (3)$$

Соотношение (1) отражает тот факт, что преобразованный автомат визуализации формирует статические иллюстрации, совпадающие с иллюстрациями исходного автомата. Соотношения (2, 3) показывают, что формирование динамических (анимационных) иллюстраций в состоянии *state* происходит таким образом, что начальная анимационная иллюстрация совпадает со статической иллюстрацией в предыдущем состоянии автомата, а конечная — со статической иллюстрацией в состоянии *state*.

2.6. Введение новых этапов в технологию

Из изложенного следует, что «рисовальщик» в визуализаторе в общем случае будет реализовываться посредством не одного, а двух операторов *switch*. При этом первый оператор *switch* будет соответствовать статическим иллюстрациям, а второй — динамическим.

Таким образом, первые четыре перечисленных шага вводятся в указанную выше технологию между этапами с номерами девять и десять, а пятый шаг – между этапами одиннадцать и двенадцать. При этом этапы технологии с первого по девятый не изменяются, а последующие шаги преобразуются следующим образом.

10. **Выбор состояний, в которых будет выполняться анимация.**
11. **Замена каждого из выбранных состояний тремя состояниями**.
12. **Использование автомата анимации**.
13. **Обеспечение взаимодействия между преобразованным и анимационным автоматами.**
14. **Выбор интерфейса визуализатора.**
15. **Сопоставление иллюстраций и комментариев с состояниями автомата**.
16. **Обеспечение выбора в каждом анимационном состоянии статического либо динамического изображения**, зависящего не только от состояния основного автомата, но и от номера шага анимации**.**
17. **Архитектура программы визуализатора**.
18. **Программная реализация визуализатора.**

3. ПОСТРОЕНИЕ ВИЗУАЛИЗАТОРА ДЛЯ АЛГОРИТМА ПИРАМИДАЛЬНОЙ СОРТИРОВКИ

Продемонстрируем технологию на примере построения визуализатора алгоритма пирамидальной сортировки. Этот алгоритм является решением общей задачи внутренней сортировки массива.

3.1. Постановка задачи сортировки (в общем виде)

Задача сортировки состоит в упорядочивании последовательности записей таким образом, чтобы значения ключевого поля составляли неубывающую последовательность [7]. В настоящей работе рассматривается более конкретный случай сортировки массива целых чисел. Задан массив a, состоящий из n элементов. Требуется реализовать алгоритм, который располагает элементы массива a в неубывающем порядке.

3.2. Решение задачи на основе алгоритма пирамидальной сортировки

Приведем словесную формулировку решения этой задачи.

Суть алгоритма пирамидальной сортировки заключается в построении и поддержании так называемой «кучи» или «пирамиды» [6]. «Пирамида» – это двоичное дерево, значения элементов в котором следуют правилу: значение родительского элемента больше или равно значению обоих дочерних элементов.

Основной механизм для поддержки и построения «пирамиды» является «погружение»:

- элемент «пирамиды» сравнивается с обоими дочерними элементами;
- если один или оба элемента оказываются больше чем вершина, то вершина обменивается с большим из дочерних элементов;
- механизм обмена осуществляется до тех пор, пока текущая вершина не окажется больше, чем элементы в обеих дочерних вершинах.

До начала решения массив представляется в виде двоичного дерева. При этом предполагается, что элемент с номером i в качестве левого и правого дочерних узлов в дереве имеет элементы с номерами $2i + 1$ и $2i + 2$.

Решение задачи предлагается строить в два этапа:

- **на первом этапе** осуществляется первичное построение «пирамиды». Для этого осуществляется последовательное «погружение» всех элементов, начиная с элемента с номером $n / 2$ до первого включительно;
- **на втором этапе** все элементы последовательно исключаются из «пирамиды» путем перемещения наибольшего элемента, находящего в вершине «пирамиды», за ее пределы. При этом на освободившееся место наибольшего элемента перемещается элемент из конца «пирамиды». Другими словами, если *hsize* — это текущий размер «пирамиды», то шаг состоит из уменьшения значения переменной *hsize* на единицу и обмена элементов с номерами 0 и *hsize*. В конце шага для поддержания «пирамиды» элемент с вершины «погружается». По окончании $(n-1)$–го шага из «пирамиды» будут исключены все элементы кроме одного, а элементы массива будут расположены в неубывающем порядке.

3.3. Выбор визуализируемых переменных

Предлагается визуализировать следующие переменные:

- *array* — массив элементов;
- *c* — текущая вершина;
- *cmax* — вершина с большим элементом при сравнениях;
- *hsize* — количество элементов в массиве, которые входят в «пирамиду» на данном шаге алгоритма.

3.4. Выбор алгоритма визуализации

Предлагается визуализировать следующие особенности алгоритма для обеспечения возможности наилучшего разъяснения его действия:

- процесс сравнения вершины с обоими дочерними;
- обмены элементов;
- оставшаяся часть «пирамиды».

3.5. Алгоритм решения задачи

Приведем алгоритм решения задачи на псевдокоде [6]:

```
1   hsize ← длина a
2   usize ← длина a / 2
3   while true do
4       if usize > 0 then
5           usize ← usize – 1
6       else if hsize > 1 then
7           hsize ← hsize – 1
8           обменять элементы 0 и hsize
9       else break
10      c = usize
11      while true do
12          cmax = наибольший из c, left(c), right(c)
13          if cmax != c then
14              обменять элементы c и cmax
15              c = cmax
16          else break
17  return результат
```

3.6. Реализация алгоритма на языке *Java*

Перепишем программу, записанную на псевдокоде, с помощью языка *Java*.

```
/**
 * Реализация алгоритма пирамидальной сортировки.
 */
private void solve() {
    // Размер пирамиды в рамках массива
    hsize = a.length;
    // Количество вершин, которые надо "погрузить"
    usize = a.length / 2;
    while (true) {
        if (usize > 0) {
            // Первый этап - упорядочивание
            usize--;
        } else if (hsize > 1){
            // Второй этап - обмен вершины пирамиды с последним элементом
```

```
            // и уменьшение размера пирамиды.
            hsize--;
            swap(0, hsize);
        } else {
            // Конец. Все вершины отсортированы
            break;
        }
        // Начинаем с usize:
        // - на первом этапе - последний "непросеянный" элемент
        // - на втором этапе - всегда вершина (usize = 0)
        int c = usize;
        while (true) {
            // "Погружение". Выбираем максимум из элемента и дочерних
            int cmax = max(c, left(c), right(c));
            // Если элемент не является максивальным
            if (cmax != c) {
                // то обмениваем его с максимумом и продолжаем "погружение"
                swap(c, cmax);
                c = cmax;
            } else {
                // иначе переходим к следующему шагу
                break;
            }
        }
    }
}
```

В этой программе операции ввода/вывода не приведены, так как их будет выполнять визуализатор. Простейшие операции `swap()` и `max()` опущены, поскольку их реализация не является существенной.

3.7. Построение схемы алгоритма по программе

Построим по тексту программы схему алгоритма (рис. 5).

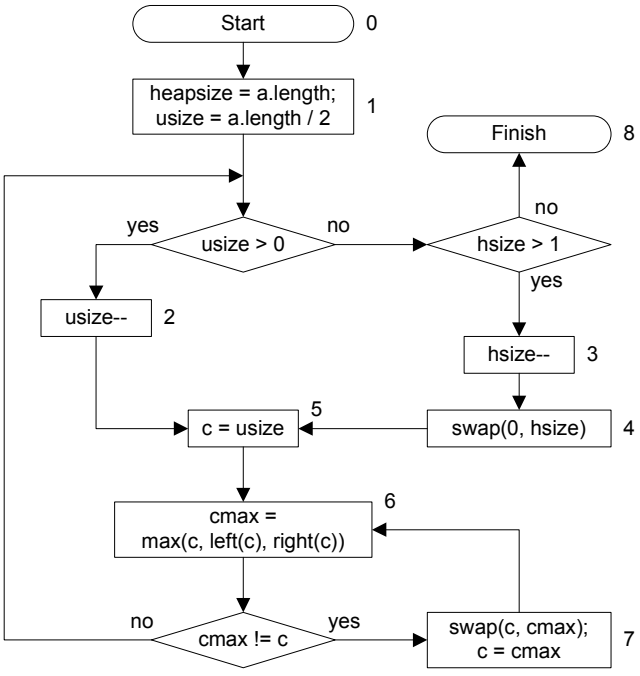

Рисунок 5. *Схема алгоритма пирамидальной сортировки*

3.8. Преобразование схемы алгоритма в автомат Мура

Следуя методу, приведенному в работе [8], построим автомат Мура, соответствующий приведенной выше схеме алгоритма (рис. 6)

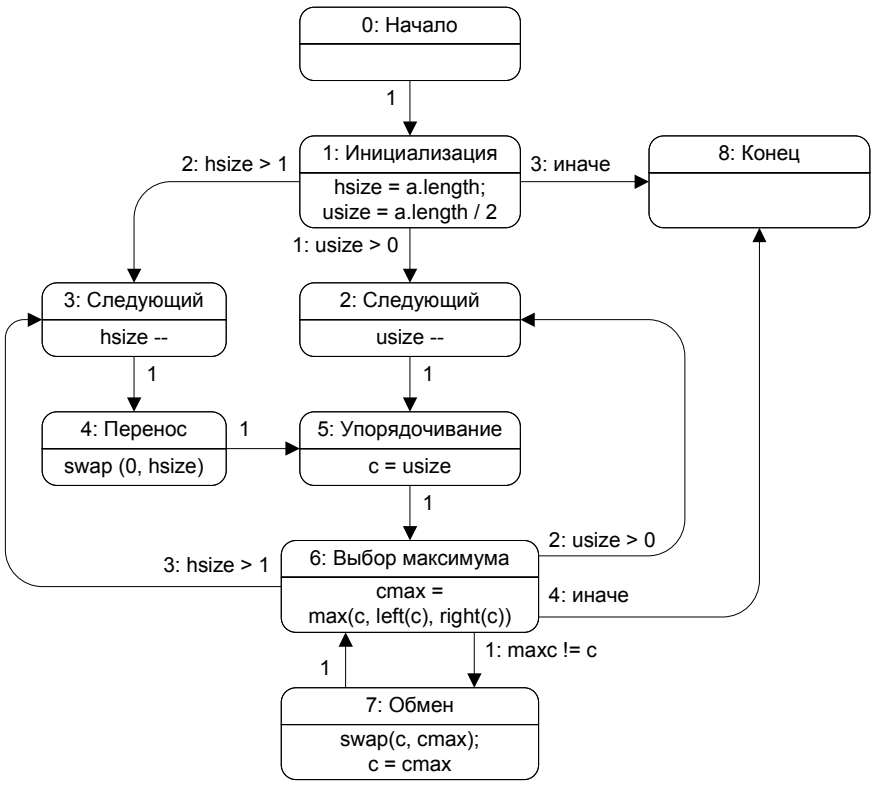

Рисунок 6. *Автомат Мура, реализующий алгоритм пирамидальной сортировки*

Исходя из схемы алгоритма на рис. 5, выделены девять состояний. В состоянии 1 осуществляется инициализация алгоритма. В состоянии 2 выполняется переход к следующему элементу на первом этапе алгоритма. В состояниях 3 и 4 элемент с вершины «пирамиды» переносится в конец и размер «пирамиды» уменьшается. Состояние 5 отвечает за выбор первой вершины для упорядочивания, в состоянии 6 выбирается максимальный элемент среди текущей вершины и дочерних, а в состоянии 7 выполняется обмен текущего элемента и максимального. Состояние 8 является конечным.

3.9. Преобразование автомата, реализующего алгоритм в автомат визуализации

В соответствии с преобразованием (рис. 1), по автомату (рис. 6) строится автомат визуализации (рис. 7). При этом состояние 2 не является существенным для визуализации и, как следствие не демонстрируется учащемуся, о чем свидетельствует отсутствие ожидания события *e0* в этом состоянии.

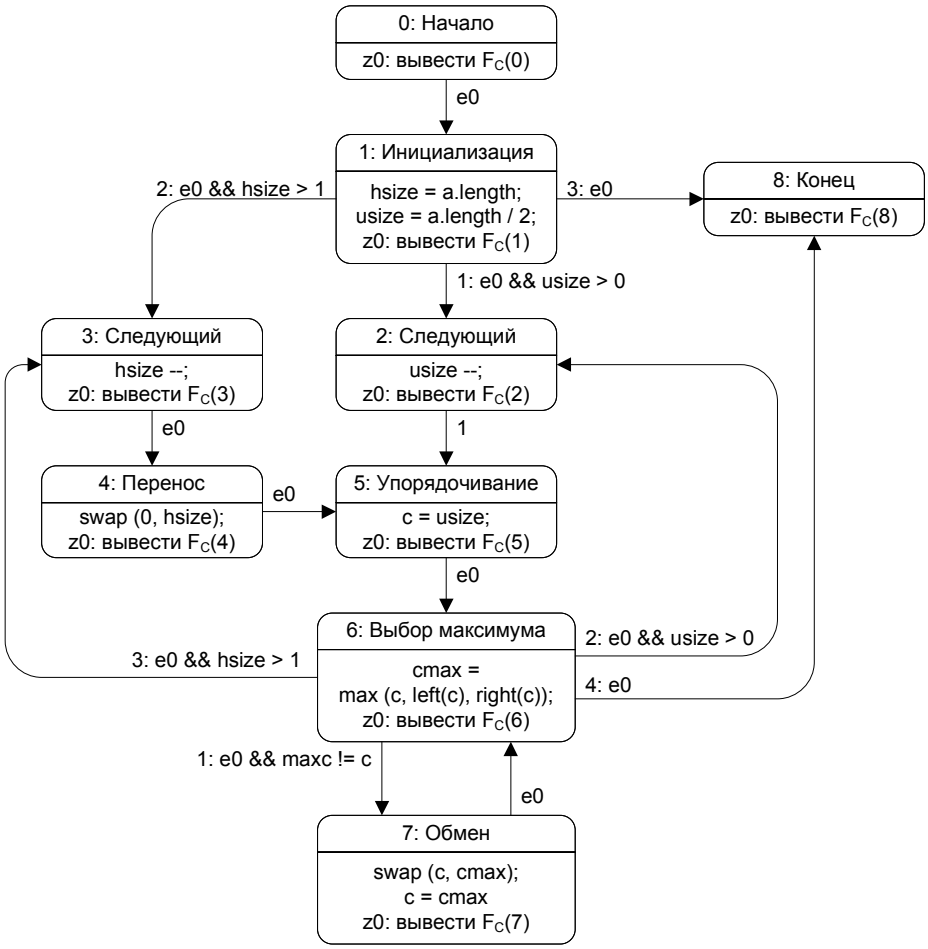

Рисунок 7. *Автомат визуализации*

3.10. Выбор состояний, в которых будет выполняться анимация

В алгоритме пирамидальной сортировки наиболее важным для динамического отображения является процесс обмена значений в вершинах, что визуально представляется как движение элементов по «пирамиде». Такими состояниями являются состояния с номерами 4 и 7.

3.11. Замена каждого из анимационных состояний тремя состояниями

В соответствии с преобразованием (рис. 2), заменим состояния 4 и 7 тройками состояний, в первом из которых запоминается номер предыдущей вершины, во втором формируется событие начала анимации, а в третьем завершается шаг визуализатора и создается конечная иллюстрация в состоянии (рис. 8). При этом событие окончания анимации формируется на переходе между анимационным и конечным состояниями шага визуализации.

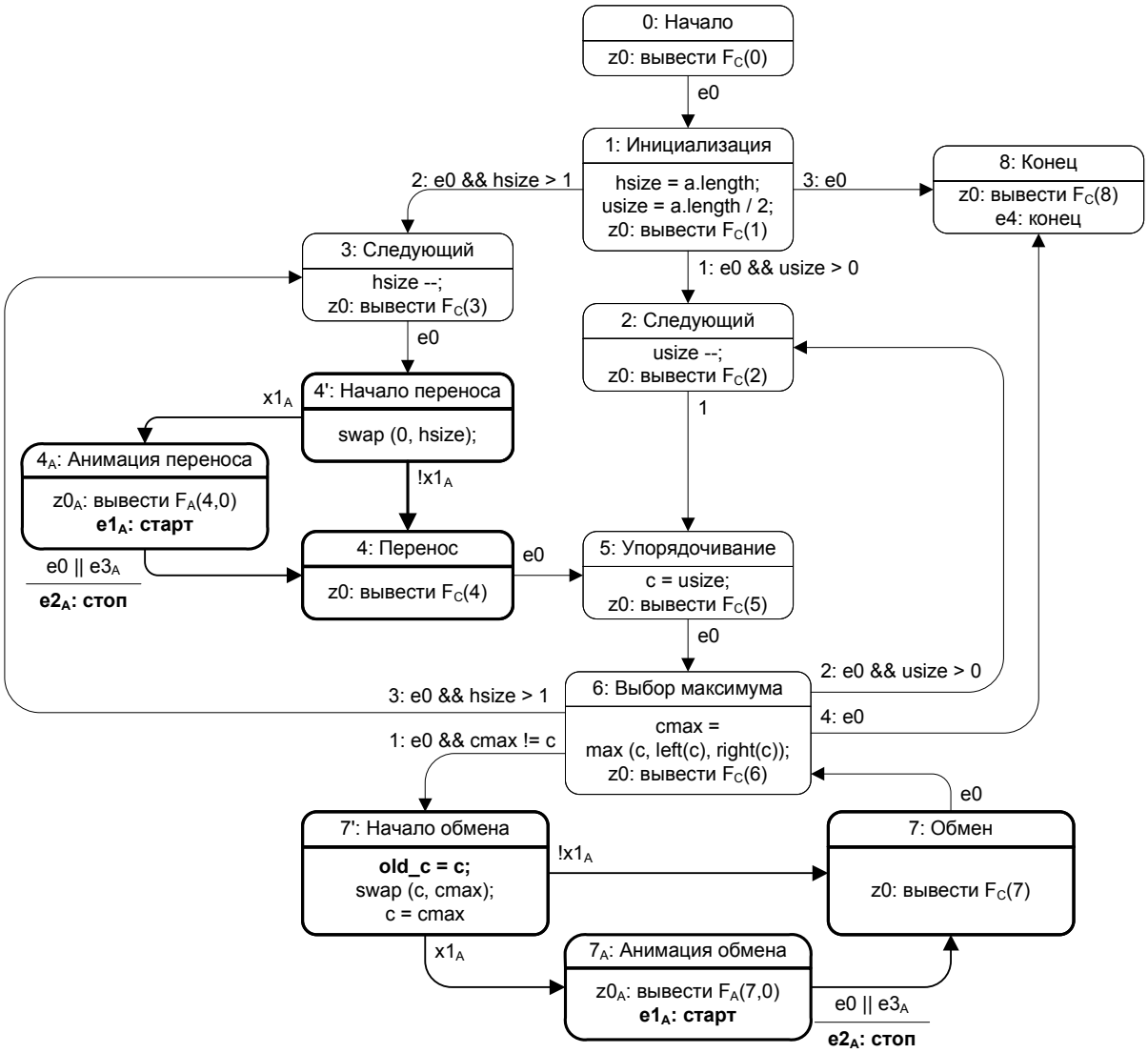

Рисунок 8. *Преобразованный автомат визуализации*

3.12. Использование автомата анимации

Как отмечалось выше, для анимации используется стандартный автомат (рис. 3).

3.13. Обеспечения взаимодействия между преобразованным и анимационным автоматами

Для обеспечения взаимодействия между автоматом визуализации и автоматом анимации вводятся связи (рис. 4).

3.14. Выбор интерфейса визуализатора

Визуализатор (рис. 9) имеет следующий интерфейс:

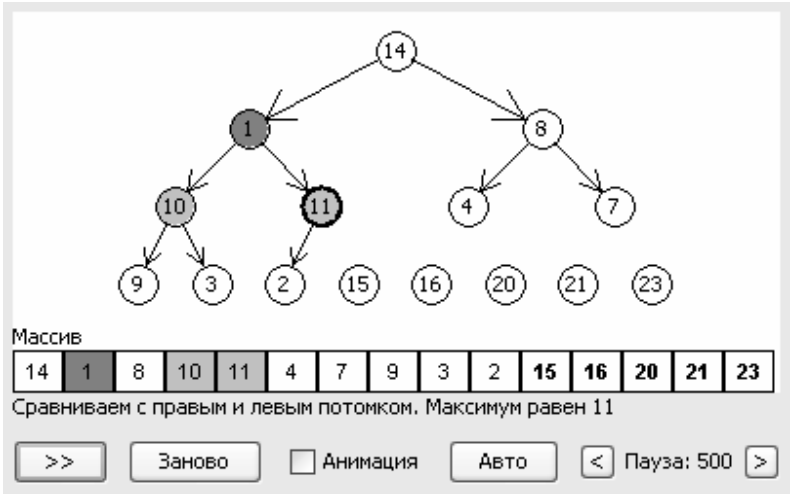

Рисунок 9. *Интерфейс визуализатора*

- в верхней части экрана изображена «пирамида». При этом ребра изображаются лишь между элементами, входящими в текущий момент в «пирамиду»;
- темным цветом отмечена текущая вершина, а серым цветом — одна или две дочерние вершины, участвующие либо в операциях сравнения, либо обмена;
- в нижней части визуализатора находится текущее состояние массива, в котором цвета ячеек повторяют цвета соответствующих элементов в «пирамиде», что помогает следить за изменениями не только в самой «пирамиде», но и в массиве, соответствующему этой «пирамиде»;
- полужирным шрифтом в массиве отмечена отсортированная часть.

Под иллюстрацией массива динамически отображаются текстовые комментарии, соответствующие текущим состояниям автомата визуализации. Например, на рис. 9 изображен комментарий, соответствующий состоянию 6. Под комментарием расположены элементы управления. Следует отметить, что по сравнению с обычным визуализатором появляется флаг управления анимацией:

- «>>» — сделать шаг алгоритма;
- «Заново» — начать алгоритм сначала;
- «Анимация» — отображать или не отображать анимацию;
- «Авто» — перейти в автоматический режим;
- «<», «>» — изменить паузу между автоматическими шагами алгоритма.

3.15. Сопоставление иллюстраций и комментариев с состояниями автомата

Для наглядности визуализации алгоритма предлагается акцентировать внимание на следующих элементах.

В состоянии 0 отображается исходный массив, расположенный в виде дерева (рис. 10).

В состоянии 1 вершины связываются в «пирамиду» посредством соединения соответствующих элементов ребрами (рис. 11).

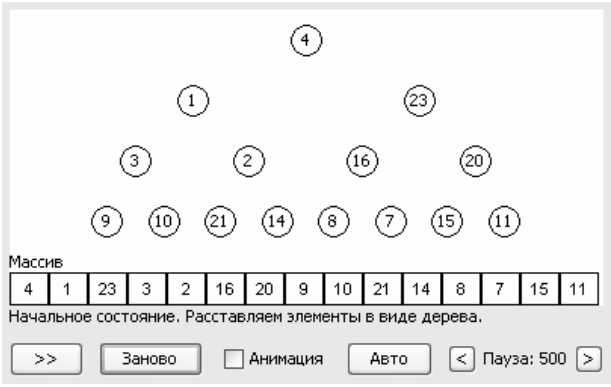

Рисунок 10. *Иллюстрация в состоянии 0*

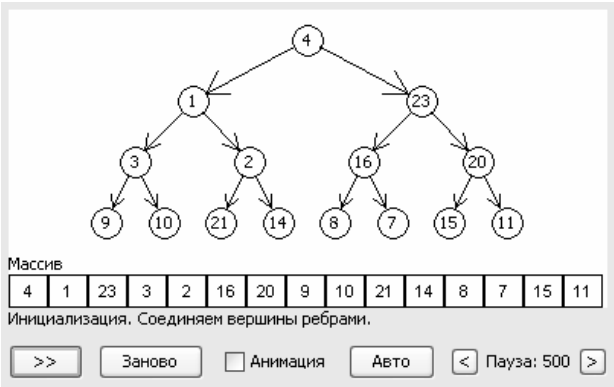

Рисунок 11. *Иллюстрация в состоянии 1*

В состоянии 3 последний элемент массива исключается из «пирамиды» (рис. 12), а в состоянии 4 происходит обмен исключенного элемента и элемента на вершине «пирамиды» (рис.13).

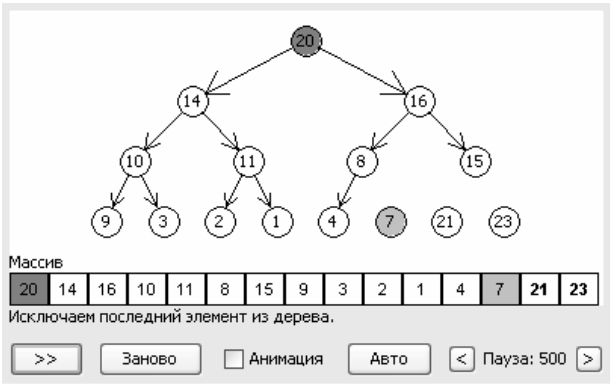

Рисунок 12. *Иллюстрация в состоянии 3*

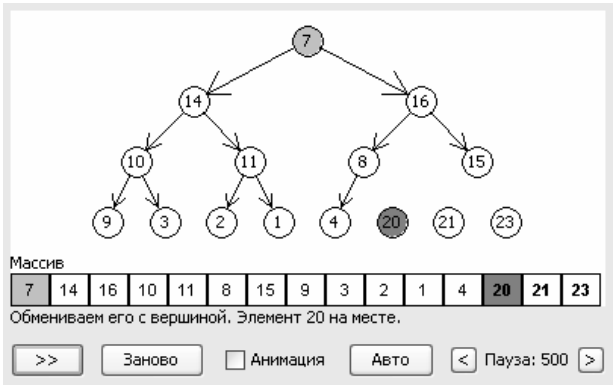

Рисунок 13. *Иллюстрация в состоянии 4*

Состояние 5 соответствует началу «погружения» вершины (рис. 14). Состояние 6 (рис. 9) соответствует процессу сравнения элемента с правым и левым дочерним элементами, а состояние 7 соответствует обмену элемента с большим из дочерних элементов (рис. 15).

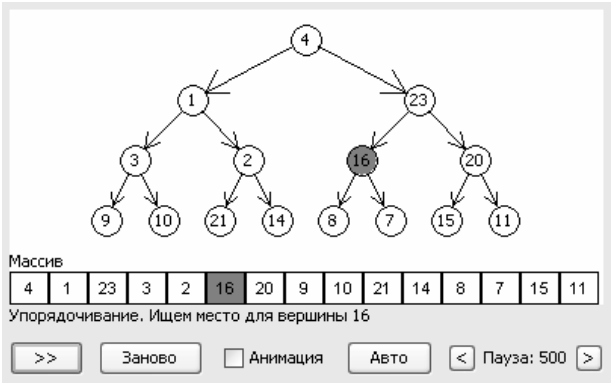

Рисунок 14. *Иллюстрация в состоянии 5*

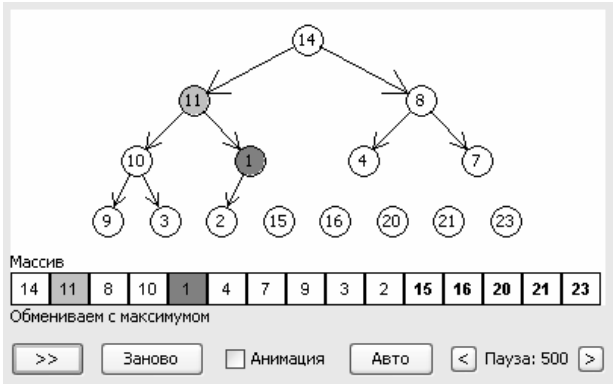

Рисунок 15. *Иллюстрация в состоянии 7*

Состояние 8 (рис. 16) является конечным и отображает отсортированный массив.

3.16. Обеспечение выбора в каждом анимационном состоянии статического или динамического изображения

Для отображения динамических иллюстраций, вершины, участвующие в обмене, отображаются на соединяющем их ребре (рис. 17).

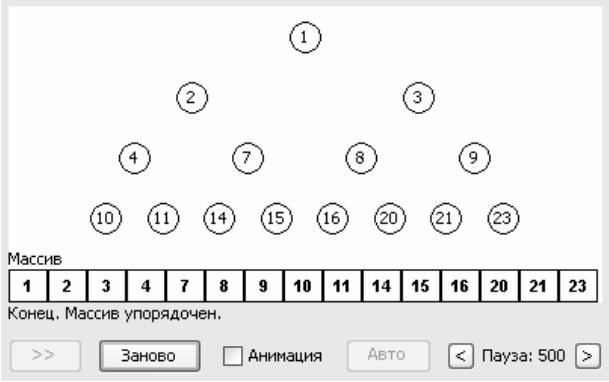

Рисунок 16. *Иллюстрация в состоянии 8*

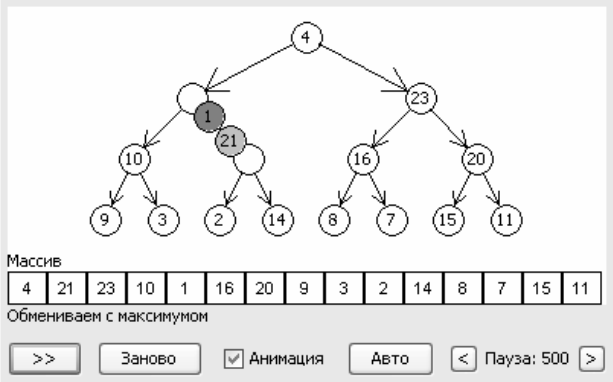

Рисунок 17. *Динамическая иллюстрация*

3.17. Архитектура программы визуализатора

Для реализации пользовательского интерфейса сформирован еще один автомат, реализующий поведение интерфейса визуализатора. Схема взаимодействия этого автомата приведена на рис. 18.

Рисунок 18. *Схема взаимодействия автомата интерфейса визуализатора*

Как следует из этого рисунка, автомат взаимодействует с автоматом визуализации и управляет интерфейсом визуализатора. Таймер используется для реализации функции *Авто*. Диаграмма переходов интерфейсного автомата приведена на рис. 19.

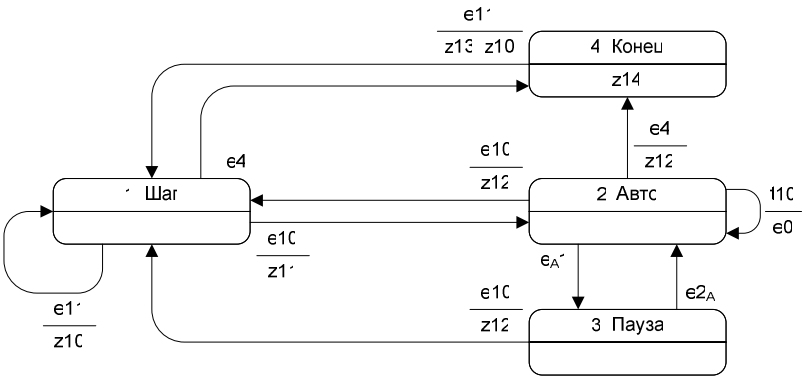

Рис. 19. *Диаграмма переходов автомата интерфейса визуализатора*

Схема взаимодействий преобразованного автомата визуализации изображена на рис. 20.

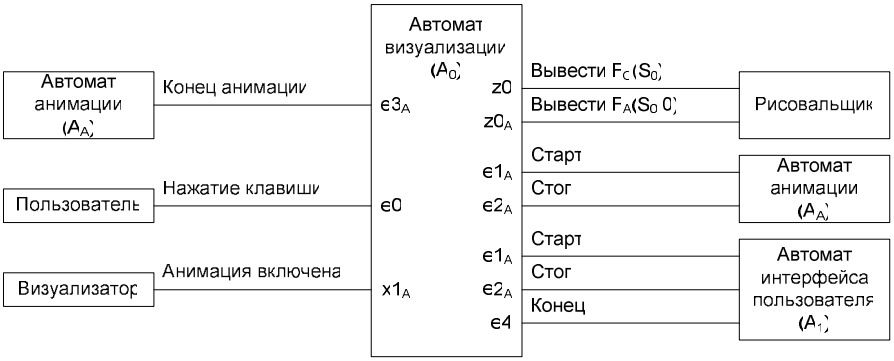

Рис. 20. *Схема взаимодействия преобразованного автомата визуализации*

Схема взаимодействия всех компонент визуализатора представлена на рис. 21.

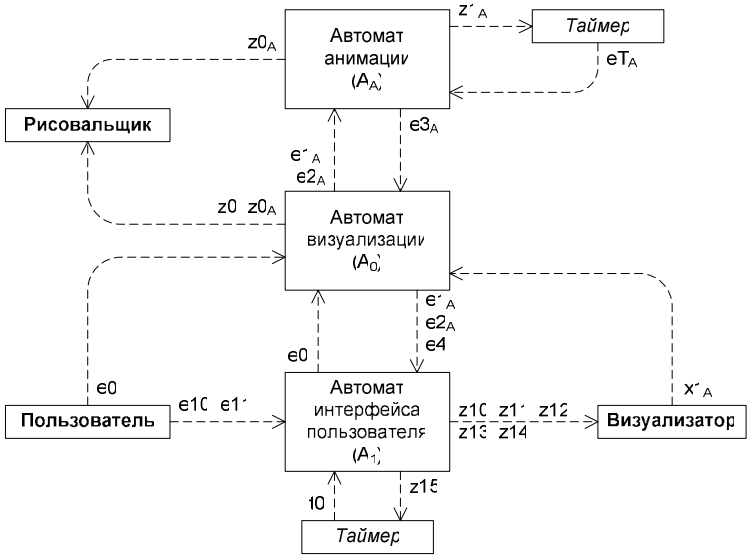

Рис. 21. *Схема взаимодействия компонент визуализатора*

3.18. Программная реализация визуализатора

Для построения визуализаторов предлагается использовать специальную библиотеку, предназначенную для реализации систем взаимодействующих автоматов. Основным принципом, заложенным в библиотеку, является прямое изоморфное преобразование схем автоматов Мили/Мура и схем их взаимодействия в специальные классы. При этом вся прикладная работа по запуску и взаимодействию автоматов осуществляется этой библиотекой. Преимуществом библиотеки является возможность реализации, в том числе, и распределенной системы автоматов.

Исходные коды и пример визуализатора приведены на сайте http://is.ifmo.ru в разделе «Статьи».

ЗАКЛЮЧЕНИЕ

В статье предложен **формализованный** подход к реализации визуализаторов алгоритмов с анимацией, улучшающий восприятие алгоритмов обучающимися, что особенно важно при дистанционном обучении. Он расширяет технологию построения визуализаторов алгоритмов на основе автоматного подхода, предложенную в работе [5].

ЛИТЕРАТУРА

1. Интернет-школа программирования. http://ips.ifmo.ru
2. **Казаков М.А., Мельничук О.П., Парфенов В.Г.** Интернет школа программирования в СПбГИТМО (ТУ). Реализация и внедрение // Материалы Всероссийская научно-методическая конференция «Телематика'2002». СПб.: 2002, с.308–309. http://tm.ifmo.ru/tm2002/db/doc/get_thes.php?id=170.
3. **Столяр С.Е., Осипова Т.Г., Крылов И.П., Столяр С.С.** Об инструментальных средствах для курса информатики // Труды II Всероссийской конференции «Компьютеры в образовании». СПб.: 1994, с.18–19.
4. **Казаков М.А., Столяр С.Е.** Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования // Материалы международной научно-методическая конференция «Телематика'2000» . СПб.: 2000, с.189–191.
5. **Казаков М.А., Шалыто А.А.** Использование автоматного программирования для реализации визуализаторов // Компьютерные инструменты в образовании. СПб.: 2004. № 2, с.19–33. http://is.ifmo.ru/works/art_vis/
6. **Кормен Т., Лайзерсон Ч., Ривест Р.** Алгоритмы. Построение и анализ. М.: МЦМНО, 2000. – 960 с.
7. **Ахо А., Хопкрофт Д., Ульман Д.** Структуры данных и алгоритмы. М.: «Вильямс», 2000. – 384 с.
8. **Шалыто А.А., Туккель Н.И.** Преобразование итеративных алгоритмов в автоматные // Программирование. 2002. № 5, с.12–26. http://is.ifmo.ru/works/iter/
9. **Шалыто А.А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. – 628 с. http://is.ifmo.ru/books/switch/1