Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А. Н. Котов, А. А. Шалыто

Построение визуализатора алгоритма Штрассена умножения матриц на базе технологии Vizi

Программирование с явным выделением состояний
Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2004

Содержание

Введение

На кафедре «Компьютерные технологии» СПбГУ ИТМО для разработки и реализации визуализаторов алгоритмов на основе конечных автоматов предложена технология *Vizi* [1, 2].

Визуализатор — это программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных.

Алгоритм Штрассена [3] является простейшим методом умножения двух матриц размера $n \times n$ с меньшей, чем классическая $\Theta(n^3)$, трудоемкостью.

В данной работе строится логика и реализация визуализатора алгоритма Штрассена умножения матриц на базе технологии *Vizi*. Сам визуализатор доступен на сайте http://is.ifmo.ru, раздел «Визуализаторы».

1. Описание и анализ алгоритма Штрассена

Матрица размера $m \times n$ элементов из некоторого пространства S — это объект из пространства $S^{m \times n}$ с упорядоченными по строкам и столбцам элементами [4]. Будут рассматриваться вещественные квадратные матрицы. Для простоты их можно считать таблицами чисел размера $n \times n$, а элементы матриц, по аналогии, называть ячейками.

Произведением $C = AB$ двух таких матриц A и B называется матрица $C = (c_{ij})$ размера $n \times n$, в которой $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$. Классический алгоритм вычисляет произведение матриц в соответствии с приведенной формулой и работает за время $\Theta(n^3) = \Theta(n^{\log 8})$.

Для перемножения матриц существует еще один простой алгоритм. Он работает рекурсивно и разбивает каждую из перемножаемых матриц на четыре части таким образом, что произведение $C = AB$ представимо в виде:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} \qquad \begin{aligned} r &= ae + bf & t &= ce + df \\ s &= ag + bh & u &= cg + dh \end{aligned}$$

На каждом шаге рекурсии происходит восемь умножений матриц размера $n/2 \times n/2$. Поэтому такой метод тоже работает за время $\Theta(n^{\log 8})$ [3].

Алгоритм Штрассена, демонстрируемый в визуализаторе, реализует схему, при которой в рекурсивном алгоритме можно обойтись семью умножениями матриц размера $n/2$ вместо восьми, сократив трудоемкость до $\Theta(n^{\log 7}) = \Theta(n^{2.81})$. Его применение оправдано для больших плотных (содержащих мало нулей) матриц.

Перед тем, как начать вычисление произведения, алгоритм проверяет, является ли размер перемножаемых матриц n натуральной степенью числа два. Если нет, то исходные матрицы следует дополнить до такого размера единицами на главной диагонали и нулями в остальных ячейках. При этом получается размер, удобный для рекурсивного вычисления, но из-за дополнительных умножений теряется эффективность метода. Например, чтобы вычислить произведение матриц размера 5×5, потребуется дополнить их до матриц размера 8×8. Использование метода Штрассена в данном случае неоправданно, поскольку $8^{2.81} > 5^3$. Но уже для всех $n > 2^{\log_{8/7} 7} \approx 30000$ классический способ умножения проигрывает в трудоемкости рассматриваемому методу.

Алгоритм Штрассена умножает две матрицы A и B размера $n \times n$ следующим образом:

1. Каждая из матриц A и B разбивается на четыре блока по схеме, приведенной выше.
2. Строятся 14 матриц $A_1, B_1, A_2, B_2, \ldots, A_7, B_7$ размера $(n/2 \times n/2)$ (для чего необходимо $\Theta(n^2)$ операций сложения/вычитания чисел).
3. Рекурсивно вычисляются семь произведений матриц меньшего размера $P_i = A_i B_i \ \ (i = 1, \ldots, 7)$.
4. Вычисляются части r, s, t, u искомой матрицы C. Они являются линейными комбинациями матриц P_i с коэффициентами из множества $\{-1, 0, 1\}$, и вычисление их требует $\Theta(n^2)$ операций сложения/вычитания чисел.

Формулы, по которым происходят вычисления на шагах 2-4, приведены в табл. 1.

Таблица 1. Формулы для алгоритма Штрассена

i	A_i	B_i	$P_i = A_i B_i$	
1	a	$g - h$	$ag - ah$	
2	$a + b$	H	$ah + bh$	$r = P_5 + P_4 - P_2 + P_6 = ae + bf$
3	$c + d$	e	$ce + de$	$s = P_1 + P_2 = ag + bh$
4	d	$f - e$	$df - de$	$t = P_3 + P_4 = ce + df$
5	$a + d$	$e + h$	$ae + ah + de + dh$	$u = P_5 + P_1 - P_3 - P_7 = cg + dh$
6	$b - d$	$f + h$	$bf + bh - df - dh$	
7	$a - c$	$e + g$	$ae + ag - ce - cg$	

Из известных алгоритмов умножения матриц асимптотически наиболее быстрый опубликован Д. Копперсмитом и С. Виноградом в работе [5]. Он позволяет перемножать квадратные матрицы за время $\Theta(n^{2.38})$, но на практике алгоритм Штрассена гораздо проще запрограммировать, и он имеет меньшую константу в оценке трудоемкости.

Дополнительные материалы по теме «Матрицы и действия с ними» можно найти в работе [4], а исчерпывающая информация о рассматриваемом алгоритме содержится в книге [3]. Кроме подробного описания, в ней также объяснено, каким образом можно прийти к такому способу умножения матриц.

2. Концепция визуализатора

2.1. Описание визуализатора

Визуализатор алгоритма Штрассена умножения матриц на базе технологии *Vizi* представляет собой *Java*-апплет. Это означает, что он может быть встроен в любой *HTML*-документ и запущен из-под любого *Web*-браузера, поддерживающего *Java*-апплеты. Если браузер (*Microsoft Internet Explorer*, *Opera*, *Mozilla*) по какой-либо причине не отображает *Java*-апплеты, то необходимо скачать для него виртуальную машину *Java* (http://java.sun.com). Визуализатор размещен по адресу http://is.ifmo.ru/, раздел «Визуализаторы».

При разработке визуализатора было принято решение выделить в качестве шагов явно только семь умножений матриц размера $n/2 \times n/2$ на первом шаге рекурсии и не заниматься визуализацией промежуточных рекурсивных умножений. С одной стороны, это позволяет сосредоточиться на идее одного шага рекурсии (первого), а с другой – существенно упрощает написание визуализатора, где остается один несложный цикл из семи умножений.

Приведем схему шагов визуализации в соответствии с описанным алгоритмом:

- исходные матрицы;
- увеличение размера матриц до 2^n;
- получение семи промежуточных матриц (семь шагов);
- получение матрицы результата;
- «обрезка» матриц до исходного размера;
- конечное состояние.

Визуализатор обеспечивает умножение двух целочисленных матриц размера до 8×8 включительно, с возможными значениями ячеек от −20 до 20. Однако эти ограничения поддаются настройке в конфигурационном файле (разд. 3.3). При запуске визуализатора на экране отображаются две перемножаемые матрицы размера 4×4, заполненные случайными элементами. Размеры и значения элементов матриц можно изменять непосредственно в запущенном визуализаторе.

Поддержка матриц размера 1×1 позволяет использовать визуализатор как инструмент для обучения детей умножению целых чисел. В таком режиме существуют только два шага визуализатора: исходный, вида “`5 × 8 = ?`”, и конечный, вида “`5 × 8 = 40`”.

2.2. Описание интерфейса визуализатора

Данный визуализатор может находиться в одном из двух режимов работы — режиме демонстрации работы алгоритма (режим показа) или режиме редактирования умножаемых матриц.

Скриншот визуализатора (в режиме демонстрации работы алгоритма) представлен на рис. 1. Здесь вычисляется произведение матриц размера 4×4, составленных из случайных элементов, и визуализируется состояние алгоритма в момент шестого из семи промежуточных умножений матриц размера 2×2.

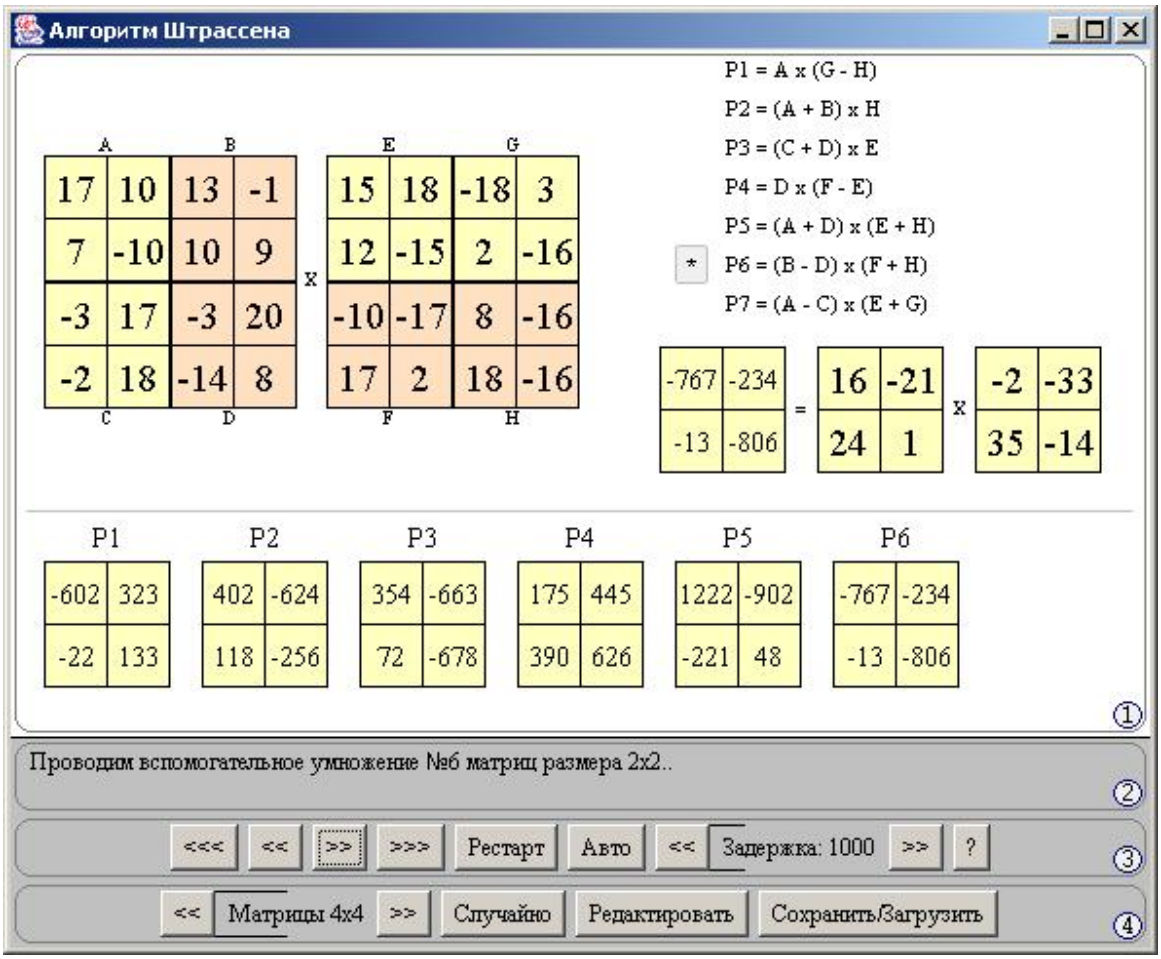

Рис. 1. Типичный вид визуализатора в режиме показа

Основные элементы пользовательского интерфейса выделены и пронумерованы.

1. *Область визуализации.* Здесь можно наблюдать состояние алгоритма Штрассена умножения матриц на текущем шаге:
   - в левой верхней части экрана расположены исходные перемножаемые матрицы (цветом выделены те части исходных матриц, которые на текущем шаге используются для получения перемножаемых матриц);
   - в нижней половине этой области визуализировано текущее состояние стека матриц алгоритма Штрассена;
   - в правой верхней части области – содержательная часть текущего шага алгоритма.
2. *Область комментариев.* В нескольких строках этой части окна отображаются комментарии к тому, что делает алгоритм на текущем шаге.
3. *Стандартная панель управления.* При помощи кнопок "<<" и ">>" в левой части можно переходить к следующему шагу алгоритма или возвращаться к предыдущему. Кнопки "<<<" и ">>>" переводят алгоритм в начальное и конечное состояния, на случай, если необходимо сразу получить результат умножения. Кнопка "Рестарт" позволяет начать все с самого начала. Кнопка "Авто" запускает визуализацию в автоматическом режиме. Пауза между шагами выставляется элементом управления "Задержка". Кнопка "?" выводит краткую информацию об авторах и используемой технологии.
4. *Панель управления матрицами.* Позволяет:
   - изменять размер перемножаемых матриц (элемент управления "Матрицы");
   - заполнить перемножаемые матрицы случайными элементами (кнопка "Случайно");
   - сохранять и загружать состояния алгоритма Штрассена, вызвав диалог сохранения состояния (кнопка "Сохранить/Загрузить");
   - переходить в режим редактирования матриц и обратно (кнопка "Редактировать").

В режиме редактирования матриц (рис. 2) область 1 визуализатора представляет собой две исходные перемножаемые матрицы, а функциональность других областей не изменяется. Для того чтобы изменить значение какой-либо ячейки матрицы, следует щелкнуть на ней левой кнопкой мыши. Выбранная ячейка будет выделена цветом и доступна для изменения с помощью клавиатуры. Можно использовать цифровые клавиши, "–", <Backspace>, <Enter> и <Esc>.

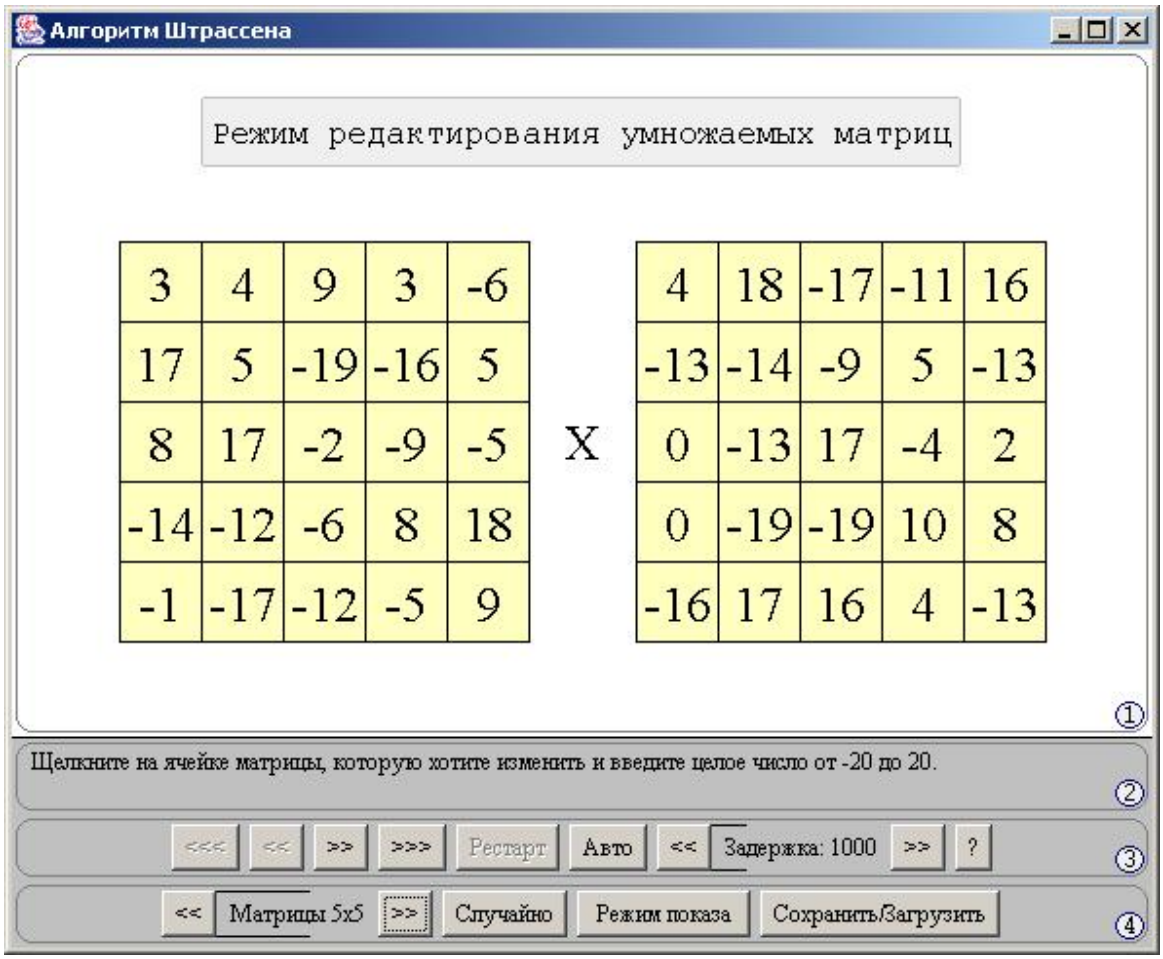

Рис. 2. Типичный вид визуализатора в режиме редактирования

2.3. Компиляция визуализатора

Среда разработки визуализатора подразумевает наличие следующих компонентов:

- операционная система — *Windows 2000*, *Windows XP* или *Windows 2003*. (http://www.microsoft.com);
- компилятор языка *Java* — *JSDK* версии 1.4.2. (http://java.sun.com);
- библиотека *Vizi* версии 0.4b5 [2];
- классы для совместимости с *MS JVM* — `classes-1.1.8.jar` [3];
- кроссплатформенный конструктор *Apache Ant* (http://ctddev.ifmo.ru/vizi/vizi-0_3-bin.rar, http://ctddev.ifmo.ru/vizi/vizi-0_4b1-bin.rar);
- исходные коды визуализатора (http://ips.ifmo.ru, раздел "Визуализаторы", или приложения 1, 2, 3);

Также необходимо установить значение переменной окружения `java_home`, равное "пути к папке", в которую установлен компилятор языка *Java* (например, “`c:\jsdk`”), и сгруппировать последние четыре компонента в структуру каталогов, о которой рассказано в документации к *Vizi* [2]. Там же приведено описание, достаточное для компиляции и запуска визуализатора.

3. Реализация визуализируемого алгоритма

Программирование визуализатора производилось в соответствии со следующим планом.

1. Разработка вспомогательных классов для структур данных, используемых алгоритмом;
2. Запись модели данных и алгоритма сразу в формате *XML+Java*, базовом для *Vizi*;
3. Разработка интерфейса визуализатора.

Рассмотрим шаги данного плана более подробно.

1. Для хранения и обработки данных в алгоритме удобно ввести два класса – `Matrix` и `Cell`, которые представляют соответственно матрицы и их отдельные ячейки. Исходные тексты этих классов приведены в Приложении 1 и хранятся в отдельных файлах `Matrix.java` и `Cell.java`. Кроме удобного математического интерфейса, в эти классы на шаге 3 был включен код, отвечающий за визуализацию матриц, что существенно упростило исходные тексты класса `StrassenVisualizer`. Это базовый класс визуализации – наследник класса `Base`, отвечающего за представление визуализатора в технологии *Vizi* [2].
2. Этот шаг предусматривает кодирование алгоритма и используемой им модели данных непосредственно в формате *XML+Java*, без создания промежуточного кода только на языке *Java*. Это достаточно спорное решение, однако оно позволяет существенно сэкономить время разработки. Текст алгоритма в формате *XML+Java* приведен в Приложении 2.
3. Процесс разработки интерфейса визуализатора (разд. 2.2) кратко можно описать так:
   - выбор примерного планируемого отображения алгоритма;
   - добавление соответствующих методов в классы `StrassenVisualizer`, `Matrix` и `Cell` (Приложение 1).

3.1. Описание модели данных

Моделью данных называется класс, содержащий все, необходимые алгоритму, переменные и структуры данных. Для реализации алгоритма Штрассена умножения матриц требуется модель данных, содержащая:

- максимальный размер матриц `int maxSize;`
- размер подготовленных матриц `int n;`
- исходный размер матриц `int l;`
- номер текущего умножения (1..7) `int rule;`
- первая исходная умножаемая матрица `Matrix A;`
- вторая исходная умножаемая матрица `Matrix B;`
- первая промежуточная умножаемая матрица `Matrix currentA;`
- вторая промежуточная умножаемая матрица `Matrix currentB;`
- матрица результата умножения `Matrix R;`
- семь промежуточных матриц `Matrix P[];`
- экземпляр апплета `StrassenVisualizer applet;`
- текущий режим работы апплета `boolean editMode;`
- текущий шаг алгоритма `int step.`

3.2. Описание программы как конечного автомата

В Приложении 2 приведен текст файла `Strassen-Algorithm.xml` на языке *XML+Java*, реализующей алгоритм Штрассена, из которой выделена модель данных и которая содержит только следующие конструкции:

- операторы присваивания;
- последовательности операторов (составные операторы);
- укороченные операторы ветвления (*if-then*);
- циклы с предусловием (*while*).

В таком виде программа может быть легко преобразована в систему конечных автоматов, как это описано в работе [1]. Технология *Vizi* генерирует на основе этого файла *java*-код, который реализует автоматы, управляющие визуализатором (Приложение 3). Полученный таким образом файл `Strassen.java` представляет собой *java*-описание алгоритма Штрассена в виде двух конечных автоматов (прямого и обратного), содержащих по $S = 17$ состояний. При этом отметим, что состояния этих автоматов совпадают. Основное преимущество автоматного

подхода – простота реализации, гарантирующая минимум отладки. Это достигается за счет формализации “привязки” изображений и комментариев визуализатора к состояниям автомата. Отметим также, что число пар “изображение, комментарий“ не превосходит числа состояний S.

3.3. Описание конфигурации визуализатора

В Приложении 1 приведен исходный код интерфейса визуализатора, написанный вручную на языке *Java* с использованием библиотеки *Vizi.* Для конфигурации внешнего вида интерфейса применяются параметры, задаваемые в файле *Strassen-Configuration.xml* (Приложение 2). Все параметры конфигурации указываются внутри тега *configuration*. Эти параметры можно условно разделить на шесть частей, приведенных в табл. 2.

Таблица 2. Параметры конфигурации визуализатора

Название параметра	Описание параметра
Параметры визуализируемых матриц	
maxMatrixSize	максимально возможный размер матриц
defaultMatrixSize	размер матриц по умолчанию
MaxABSFillValue	максимально возможный модуль значения ячеек матриц, от которого зависит количество цифр в ячейках промежуточных матриц
Строки, отображаемые в режиме редактирования	
editModeHeader	заголовок режима редактирования
editModeComment	комментарий к этому режиму
Цветовая гамма визуализатора	
matrix-style	шесть возможных схем отображения элементов визуализатора
Параметры области комментариев	
comment-height	высота области для комментариев
Параметры панели управления матрицами	
size	панель изменения размера перемножаемых матриц
button-random	кнопка заполнения матриц случайными элементами
button-edit, button-show	кнопка переключения визуализатора в режим показа и режим редактирования — фактически реализует еще один автомат с двумя состояниями
Параметры окна сохранения/загрузки состояния	

SaveLoadDialog	конфигурация окна сохранения/загрузки состояния
loadStateHint	подсказка с описанием формата сохраняемого состояния
incorrectState1	подсказка с описанием причины неудачного сохранения состояния

Заключение

Технология *Vizi* является очередным шагом по упрощению и автоматизации процесса проектирования визуализаторов [6]. Процесс разбивается на несколько относительно независимых друг от друга шагов, уменьшается доля эвристических решений при разработке. Творческая свобода программиста остается только на этапе написания кода, который отвечает за отображение состояний визуализатора. Остальные этапы достаточно жестко регламентированы. При этом технология *Vizi* обладает следующими преимуществами:

- разработка более качественных визуализаторов за более короткое время;
- автоматическая генерация кода “обратного” автомата.

Также отметим пользу в разделении кода на общий для всех визуализаторов и разрабатываемый индивидуально для конкретного визуализатора. Они компилируются отдельно и в исполняемом виде представляют собой два разных `jar`-файла. Таким образом, достигаются два существенных преимущества при организации коллекций визуализаторов:

- сокращается размер `jar`-файлов конкретных визуализаторов;
- появляется возможность обновить управляющие элементы у всей коллекции визуализаторов одновременно, с помощью обновления только одного `jar`-файла (новой версией библиотеки *Vizi*).

Последнее возможно при условии совместимости библиотеки с предыдущими версиями, заявленной в документации к *Vizi*. Эта совместимость наблюдается, начиная с версии *Vizi* 0.4.

В то же время технология *Vizi* имеет одно существенное ограничение. Она позволяет визуализировать состояния алгоритма, но не переходы между ними, которые могли бы более наглядно продемонстрировать суть алгоритма. Также это ограничение не позволяет использовать элементы анимации в визуализаторах.

В принципе, это ограничение может быть снято, если нынешний интерфейс функции рисования `draw(int state, Data data)` расширить следующим образом:

```
draw(int state, int nextState, Data data, double delay).
```

Это значительно осложнит процесс разработки визуализаторов, но зато появится средство для визуализации переходов между состояниями.

Источники

1. *Казаков М.А., Корнеев Г.А., Шалыто А.А*. Разработка логики визуализаторов алгоритмов на основе конечных автоматов //Телекоммуникации и информатизация образования. 2003, № 6, с. 27-58. http://is.ifmo.ru, раздел «Статьи».
2. *Vizi Home Page* — http://ctddev.ifmo.ru/vizi/.
3. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦМНО, 1999, с. 679-684.
4. *Котов А.* Матрицы и действия с ними — http://rain.ifmo.ru/cat/view.php/theory/unsorted/matrices-2004.
5. *Coppersmith D., Winograd S.* Matrix multiplication via arithmetic progressions //Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, 1987, p. 1- 6.
6. *Казаков М.А., Шалыто А.А*. Использование автоматного программирования для реализации визуализаторов //Компьютерные инструменты в образовании». 2004. № 2, с. 19-33. http://is.ifmo.ru, раздел «Статьи».

Приложение 1. Исходные коды интерфейса визуализатора

Файл Cell.java

```
package ru.ifmo.vizi.strassen;

import java.awt.Container;
import ru.ifmo.vizi.base.widgets.Rect;
import ru.ifmo.vizi.base.widgets.ShapeStyle;



/**
 * Cell of the Matrix class.
 *
 * @version 1.0 (22.02.2004)
 * @author  Alex Kotov
 */
public class Cell {

       /**
        * Value of cell.
        */
    private int value = 0;

       /**
        * Shape for drawing cell.
        */
    private Rect rect = null;

       /**
        * Cell rect style number.
        */
    private int currentStyle = 0;



       /**
        * Cell constructor.
        *
        * @param v value to put into each cell.
        */
       Cell(int v) {
              value = v;
       }

       /**
        * Updates value of the cell.
        *
        * @param v new value for this cell.
        */
       public void setValue(int v) {
              value = v;
              if (rect != null) rect.setMessage(v + "");
       }

       /**
        * Returns value of the cell.
        *
        * @return current value of the cell.
        */
       public int getValue() {
              return value;
```

```
	}



	/**
	 * Returns Shape of the matrix (just for matrix-cells).
	 *
	 * @return rectangular shape assotiated with the matrix.
	 */
	public Rect getRect() {
		return rect;
	}

	/**
	 * Creates shape for cell.
	 *
	 * @param styleSet style parameters for creating shape.
	 */
	public void createRectBase(ShapeStyle[] styleSet) {
		if (rect == null) rect = new Rect(styleSet, value + "");
		else rect.setMessage(value + "");
		rect.setStyle(currentStyle);
	}

	/**
	 * Set choosed style for shapes of the cell.
	 *
	 * @param styleNum number of style to set.
	 */
	public void updateRectBase(int styleNum) {
		currentStyle = styleNum;
	}

	/**
	 * Puts cell shape to the container for drawing.
	 *
	 * @param p container for cell to put.
	 * @param x1 left border for placing cell in container.
	 * @param y1 up border for placing cell in container.
	 * @param x2 right border for placing cell in container.
	 * @param y2 down border for placing cell in container.
	 * @param m string for adjusting font size in all cells.
	 */
	public void layoutCell(Container p, int x1, int y1, int x2, int y2, String m) {
		if (rect != null) {
			rect.setLocation(x1, y1);
			rect.setSize(x2 - x1 + 1, y2 - y1 + 1);
			rect.adjustFontSize(m);
			p.add(rect);
		}
	}

	/**
	 * Returns value length for the cell.
	 *
	 * @return maximum number of ciphers for the cell.
	 */
	public int getLength() {
		String valueString = "" + value;
		return valueString.length();
	}
}
```

Файл Matrix.java

```
package ru.ifmo.vizi.strassen;

import java.awt.Container;
import ru.ifmo.vizi.base.widgets.ShapeStyle;



/**
 * Matrix class.
 *
 * @version 1.0 (12.01.2004)
 * @author  Alex Kotov
 */
public class Matrix {

	/**
	 * Number of cells in matrix row.
	 */
	private int width;

	/**
	 * Number of cells in matrix col.
	 */
	private int height;

	/**
	 * Array of matrix cells.
	 */
	private Cell cell[][];

	/**
	 * Borders of matrix which was set during last drawing.
	 */
	private int x1, y1, x2, y2;



	/**
	 * Matrix constructor for creating empty matrix with choosed size.
	 *
	 * @param s size of matrix to create.
	 */
	Matrix(int w, int h) {
		width = w;
		height = h;
		cell = new Cell[width + 1][height + 1];
		for (int y = 1; y <= height; y++)
			for (int x = 1; x <= width; x++) cell[x][y] = new Cell(0);
	}

	/**
	 * Matrix constructor from Matrix parts.
	 *
	 * @param m1 NW part of new matrix.
	 * @param m2 NE part of new matrix.
	 * @param m3 SW part of new matrix.
	 * @param m4 SE part of new matrix.
	 */
	Matrix(Matrix m1, Matrix m2, Matrix m3, Matrix m4) {
		width = m1.getWidth() + m2.getWidth();
		height = m1.getHeight() + m3.getHeight();
		cell = new Cell[width + 1][height + 1];
		for (int y = 1; y <= m1.getHeight(); y++) {
```

```
                    for (int x = 1; x <= m1.getWidth(); x++) cell[x][y] = m1.getCell(x, y);
                    for (int x = m1.getWidth() + 1; x <= width; x++) cell[x][y] =
m2.getCell(x - m1.getWidth(), y);
              }
              for (int y = m1.getHeight() + 1; y <= height; y++) {
                    for (int x = 1; x <= m1.getWidth(); x++) cell[x][y] = m3.getCell(x, y -
m1.getHeight());
                    for (int x = m1.getWidth() + 1; x <= width; x++) cell[x][y] =
m4.getCell(x - m1.getWidth(), y - m1.getHeight());
              }
        }

        /**
         * Matrix constructor for creating copy of matrix
         *
         * @param m matrix to make copy from.
         */
        Matrix(Matrix m) {
              width = m.getWidth();
              height = m.getHeight();
              cell = new Cell[width + 1][height + 1];
              for (int y = 1; y <= height; y++)
                    for (int x = 1; x <= width; x++)
                          cell[x][y] = new Cell(m.getCell(x, y).getValue());
        }




        /**
         * Returns matrix width (number of row-side cells).
         *
         * @return matrix width.
         */
        public int getWidth() {
              return width;
        }

        /**
         * Returns matrix height (number of col-side cells).
         *
         * @return matrix height.
         */
        public int getHeight() {
              return height;
        }

        /**
         * Returns matrix-cell in y-th row and x-th col of the matrix.
         *
         * @param x col number of choosed cell.
         * @param y row number of choosed cell.
         * @return choosed matrix-cell.
         */
        public Cell getCell(int x, int y) {
              return cell[x][y];
        }

        /**
         * Updates cell of matrix in y-th row and x-th col.
         *
         * @param c new cell for choosed place.
         * @param x col number of choosed cell.
         * @param y row number of choosed cell.
         */
        public void setCell(Cell c, int x, int y) {
```

```
        cell[x][y] = c;
}

/**
 * Updates value of the cell of matrix in y-th row and x-th col.
 *
 * @param v new value for choosed cell.
 * @param x col number of choosed cell.
 * @param y row number of choosed cell.
 */
public void setValue(int v, int x, int y) {
        cell[x][y].setValue(v);
}

/**
 * Returns value of cell of the matrix in y-th row and x-th col.
 *
 * @param x col number of choosed cell.
 * @param y row number of choosed cell.
 * @return current value of choosed cell.
 */
public int getValue(int x, int y) {
        return cell[x][y].getValue();
}



/**
 * Returns NW part of the matrix.
 *
 * @return north-west part of the matrix.
 */
public Matrix getA() {
        return getMatrixPart(1, 1, width / 2, height / 2);
}

/**
 * Returns NE part of the matrix.
 *
 * @return north-east part of the matrix.
 */
public Matrix getB() {
        return getMatrixPart(width / 2 + 1, 1, width / 2, height / 2);
}

/**
 * Returns SW part of the matrix.
 *
 * @return south-west part of the matrix.
 */
public Matrix getC() {
        return getMatrixPart(1, height / 2 + 1, width / 2, height / 2);
}

/**
 * Returns SE part of the matrix.
 *
 * @return south-east part of the matrix.
 */
public Matrix getD() {
        return getMatrixPart(width / 2 + 1, height / 2 + 1, width / 2, height / 2);
}
```

```
/**
 * Returns part of the matrix with choosed borders.
 *
 * @param ox1 col from which to start taking part.
 * @param oy1 row from which to start taking part.
 * @param owidth width (number of cells) of choosed part.
 * @param oheight height (number of cells) of choosed part.
 * @return choosed part of the matrix.
 */
public Matrix getMatrixPart(int ox1, int oy1, int owidth, int oheight) {
      Matrix result = new Matrix(owidth, oheight);
      for (int y = 1; y <= oheight; y++) for (int x = 1; x <= owidth; x++) {
            if ((y + oy1 - 1 <= height) && (x + ox1 - 1 <= width))
                  result.setCell(cell[x + ox1 - 1][y + oy1 - 1], x, y);
            else
                  result.setValue((x == y) ? 1 : 0, x, y);
      }
      return result;
}
```

```
/**
 * Calculates sum of current matrix and matrix Q.
 *
 * @param Q matrix to calculate sum with.
 * @return new matrix with sum.
 */
public Matrix add(Matrix Q) {
      Matrix result = new Matrix(width, height);
      for (int y = 1; y <= height; y++) for (int x = 1; x <= width; x++)
        result.setValue(cell[x][y].getValue() + Q.getCell(x, y).getValue(), x, y);
      return result;
}

/**
 * Calculates difference between current matrix and matrix Q.
 *
 * @param Q matrix to calculate difference with.
 * @return new matrix with difference.
 */
public Matrix sub(Matrix Q) {
      Matrix result = new Matrix(width, height);
      for (int y = 1; y <= height; y++) for (int x = 1; x <= width; x++)
        result.setValue(cell[x][y].getValue() - Q.getCell(x, y).getValue(), x, y);
      return result;
}

/**
 * Calculates product of current matrix and matrix Q.
 *
 * @param Q matrix to calculate product with.
 * @return new matrix with product.
 */
public Matrix mul(Matrix Q) {
      Matrix result = new Matrix(width, height);
      for (int y = 1; y <= height; y++) for (int x = 1; x <= width; x++) {
            int value = 0;
            for (int k = 1; k <= width; k++)
                  value += (cell[k][y].getValue() * Q.getCell(x, k).getValue());
            result.setValue(value, x, y);
      }
      return result;
}
```

```
	/**
	 * Creates shape for all matrix cells.
	 *
	 * @param styleSet style parameters for creating shapes.
	 */
	public void createRectBase(ShapeStyle[] styleSet) {
		for (int y = 1; y <= height; y++)
			for (int x = 1; x <= width; x++)
				cell[x][y].createRectBase(styleSet);
	}

	/**
	 * Set choosed style for all shapes of the matrix.
	 *
	 * @param styleNum number of style to set.
	 */
	public void updateRectBase(int styleNum) {
		for (int y = 1; y <= height; y++)
			for (int x = 1; x <= width; x++)
				cell[x][y].updateRectBase(styleNum);
	}

	/**
	 * Puts all matrix shapes to the container for drawing.
	 *
	 * @param p container for matrix to put.
	 * @param ox1 left border for placing matrix in container.
	 * @param oy1 up border for placing matrix in container.
	 * @param ox2 right border for placing matrix in container.
	 * @param oy2 down border for placing matrix in container.
	 * @param m string for adjusting font size in all cells.
	 */
	public void layoutMatrix(Container p, int ox1, int oy1, int ox2, int oy2, String m) {
		x1 = ox1;
		y1 = oy1;
		x2 = ox2;
		y2 = oy2;

		final int sizeX = (x2 - x1) / width;
		final int sizeY = (y2 - y1) / height;
		for (int y = 1; y <= height; y++) for (int x = 1; x <= width; x++)
			cell[x][y].layoutCell(p, x1 + (x - 1) * sizeX, y1 + (y - 1) * sizeY, x1
+ x * sizeX, y1 + y * sizeY, m);
	}

	/**
	 * Checks if choosed point placed inside the matrix in paint area.
	 *
	 * @param x x-coordinate of choosed point.
	 * @param y y-coordinate of choosed point.
	 * @return cell if choosed point placed inside the matrix in paint area.
	 */
	public Cell inside(int x, int y) {
		Cell result = null;
		if ((x >= x1) && (x <= x2) && (y >= y1) && (y <= y2)) {
			x = width * (x - x1) / (x2 - x1) + 1;
			y = height * (y - y1) / (y2 - y1) + 1;
			result = getCell(x, y);
		}
		return result;
	}
```

```
	/**
	 * Returns maximum value length for all cells.
	 *
	 * @return maximum number of ciphers for all cells.
	 */
	public int getMaxLength() {
		int result = 1;
		for (int y = 1; y <= height; y++) for (int x = 1; x <= width; x++)
			result = Math.max(result, cell[x][y].getLength());
		return result;
	}
}
```

Файл StrassenVisualizer.java

```
package ru.ifmo.vizi.strassen;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.StringTokenizer;
import ru.ifmo.vizi.base.ui.AutoControlsPane;
import ru.ifmo.vizi.base.ui.AdjustablePanel;
import ru.ifmo.vizi.base.ui.HintedButton;
import ru.ifmo.vizi.base.ui.MultiButton;
import ru.ifmo.vizi.base.ui.SaveLoadDialog;
import ru.ifmo.vizi.base.widgets.Rect;
import ru.ifmo.vizi.base.widgets.ShapeStyle;
import ru.ifmo.vizi.base.Base;
import ru.ifmo.vizi.base.Configuration;
import ru.ifmo.vizi.base.I18n;
import ru.ifmo.vizi.base.SmartTokenizer;
import ru.ifmo.vizi.base.VisualizerParameters;



/**
 * StrassenVisualizer main class.
 *
 * @version 1.0 (12.01.2004)
 * @author  Alex Kotov
 */
public final class StrassenVisualizer extends Base
		implements MouseListener, KeyListener, AdjustmentListener {

	/**
	 * Strassen automata instance.
	 */
	private final Strassen auto;

	/**
	 * Strassen automata (global) data.
	 */
	private Strassen.Data data;
```

```
/**
 * Save/load dialog.
 */
private SaveLoadDialog saveLoadDialog;

/**
 * Edit/Show switch mode button.
 */
private MultiButton modeButton;

/**
 * Matrix size switch control element.
 */
private AdjustablePanel sizePanel;

/**
 * Visualizer style parameters.
 */
private final ShapeStyle[] styleSet;

/**
 * Active (editable) cell of matrix in edit mode.
 */
private Cell activeCell;

/**
 * True, if current value of activeCell is negative.
 */
private boolean negativeActiveCell;

/**
 * Current absolute value of active cell.
 */
private String handleActiveCell;

/**
 * Rules for calculation P1, P2, ..., P7.
 */
public static final String rules[] = new String[]{ "",
      "A x (G - H)",
      "(A + B) x H",
      "(C + D) x E",
      "D x (F - E)",
      "(A + D) x (E + H)",
      "(B - D) x (F + H)",
      "(A - C) x (E + G)" };

/**
 * Different global drawing parameters.
 */
private int sizeX, sizeY, gapX, gapY, sX, sY;

/**
 * Something like superFrame (ask Kotov Vladimir what is it :).
 */
private final Frame forefather;



/**
 * StrassenVisualizer constructor.
 *
 * @param parametres system parametres such as automata, locale etc.
 */
public StrassenVisualizer(VisualizerParameters parameters) {
```

```
            super(parameters);
            this.forefather = parameters.getForefather();
            auto = new Strassen(locale);
            data = auto.d;
            data.applet = this;
            data.maxSize = config.getInteger("maxMatrixSize");
            data.l = Math.min(config.getInteger("defaultMatrixSize"), data.maxSize);

            styleSet = ShapeStyle.loadStyleSet(config, "matrix-style");
            clientPane.addMouseListener(this);
            clientPane.addKeyListener(this);

            setRandomMatrixes();
            createInterface(auto);
      }


      /**
       * Creates user controls Pane.
       *
       * @return just created Pane.
       */
      public Component createControlsPane() {
            Panel panel = new Panel(new BorderLayout());
            panel.add(new AutoControlsPane(config, auto, forefather, true),
BorderLayout.NORTH);

            Panel matrixPanel = new Panel(new FlowLayout());
                  sizePanel = new AdjustablePanel(config, "size");
                  int maxActualSize = 1;
                  while (maxActualSize < data.maxSize) maxActualSize *= 2;
                  data.maxSize = maxActualSize;
                  sizePanel.setMaximum(maxActualSize);
                  sizePanel.setValue(data.l);
                  sizePanel.addAdjustmentListener(this);
            matrixPanel.add(sizePanel);
                  HintedButton randomButton = new HintedButton(config, "button-random") {
                        protected void click() {
                              setRandomMatrixes();
                        }
                  };
            matrixPanel.add(randomButton);
                  modeButton = new MultiButton(config, new String[]{"button-edit",
"button-show"}) {
                        protected int click(int state) {
                              data.editMode = !data.editMode;
                              activeCell = null;
                              data.A = data.A.getMatrixPart(1, 1, data.l, data.l);
                              data.B = data.B.getMatrixPart(1, 1, data.l, data.l);
                              resetSelect();
                              algoRestart();
                              return 1 - state;
                        }
                  };
            matrixPanel.add(modeButton);
                  HintedButton saveButton = new HintedButton(config, "button-SaveLoad") {
                  protected void click() {
                              saveLoadDialog.center();

      saveLoadDialog.setComment(config.getParameter("loadStateHint"));
                              saveLoadDialog.show(getState());
                  }
                  };
            matrixPanel.add(saveButton);
            panel.add(matrixPanel, BorderLayout.CENTER);
```

```
            saveLoadDialog = new SaveLoadDialog(config, forefather) {
                  public boolean load(String text) throws Exception {
                        int result = setState(text);
                        updateScreen();
                        if (result > 0) {
                              setComment(config.getParameter("incorrectState" +
result));
                              return false;
                        }
                        setComment(config.getParameter("loadStateHint"));
                        return true;
                  }
            };
            return panel;
      }




      /**
       * Creates string which describe current state of visualizer.
       *
       * @return string with current state description.
       */
      private String getState() {
            StringBuffer stateString = new StringBuffer();
            stateString.append("step = ");
            stateString.append(data.step);
            stateString.append(";\nsize = ");
            stateString.append(data.l);
            stateString.append(";\nA = (");
            for (int y = 1; y <= data.l; y++)
                  for (int x = 1; x <= data.l; x++) {
                        stateString.append(data.A.getValue(x, y));
                        if ((x < data.l) || (y < data.l)) stateString.append(",");
                  }
            stateString.append(");\nB = (");
            for (int y = 1; y <= data.l; y++)
                  for (int x = 1; x <= data.l; x++) {
                        stateString.append(data.B.getValue(x, y));
                        if ((x < data.l) || (y < data.l)) stateString.append(",");
                  }
            stateString.append(");");
            return stateString.toString();
      }


      /**
       * Put state of visualizer as it defined in parameter.
       *
       * @param state string with description of state to set.
       * @return successful setting state (0) or number of error.
       */
      private int setState(String state) throws Exception {
            // Parse new state
            SmartTokenizer tokenizer = new SmartTokenizer(state, config);
            tokenizer.expect("step");
            tokenizer.expect("=");
            int step = tokenizer.nextInt();
            tokenizer.expect(";");
            tokenizer.expect("size");
            tokenizer.expect("=");
```

```
        int userSize = tokenizer.nextInt();
        if (userSize > data.maxSize) return 1; // Very big matrix error
        tokenizer.expect(";");

        int A[] = new int[userSize * userSize + 1];
        int B[] = new int[userSize * userSize + 1];
        int maxABSFillValue = config.getInteger("MaxABSFillValue");
        tokenizer.expect("A");
        tokenizer.expect("=");
        tokenizer.expect("(");
        for (int i = 1; i <= userSize * userSize; i++) {
              A[i] = tokenizer.nextInt(-maxABSFillValue, maxABSFillValue);
              if (i < userSize * userSize) tokenizer.expect(",");
              else tokenizer.expect(")");
        }
        tokenizer.expect(";");
        tokenizer.expect("B");
        tokenizer.expect("=");
        tokenizer.expect("(");
        for (int i = 1; i <= userSize * userSize; i++) {
              B[i] = tokenizer.nextInt(-maxABSFillValue, maxABSFillValue);
              if (i < userSize * userSize) tokenizer.expect(",");
              else tokenizer.expect(")");
        }
        tokenizer.expect(";");
        tokenizer.expectEOF();

        // Apply new state
        data.l = userSize;
        if (sizePanel != null) sizePanel.setValue(data.l);
        data.A = new Matrix(userSize, userSize);     // Set new matrixes
        data.B = new Matrix(userSize, userSize);
        for (int y = 1; y <= userSize; y++)
              for (int x = 1; x <= userSize; x++) {
                    data.A.setValue(A[(y - 1) * userSize + x], x, y);
                    data.B.setValue(B[(y - 1) * userSize + x], x, y);
              }
        algoRestart();                                    // Set new step
        for (int i = 1; i <= step; i++) auto.stepForward(0);
        return 0;
  }


  /**
   * Randomizes source matrixes and restarts algorythm.
   */
  private void setRandomMatrixes() {
        data.A = randomMatrix();
        data.B = randomMatrix();
        algoRestart();
  }


  /**
   * Creates matrix with random values in each cell.
   *
   * @return just created matrix.
   */
  private Matrix randomMatrix() {
        final int rndMax = config.getInteger("MaxABSFillValue");
        Matrix result = new Matrix(data.l, data.l);
        for (int y = 1; y <= data.l; y++) for (int x = 1; x <= data.l; x++)
          result.setValue((int)(Math.random() * (2 * rndMax + 1)) - rndMax, x, y);
        return result;
  }
```

```
	/**
	 * Reset the whole state of 'auto' (as if applet has just restarted).
	 */
	private void algoRestart() {
		auto.toStart();
		data.step = 0;
		data.rule = 0;
		data.n = 1;
		while (data.n < data.l) data.n *= 2;
		updateScreen();
	}



	/**
	 * Redraws state of Strassen visualizer.
	 */
	public void updateScreen() {
		update(true);
		clientPane.doLayout();
	}


	/**
	 * Invoked when client pane shoud be layouted.
	 *
	 * @param clientWidth client pane width.
	 * @param clientHeight client pane height.
	 */
	protected void layoutClientPane(int width, int height) {
		final int totalGapsX, totalGapsY;
		final double gapAspectX, gapAspectY, totalMatrixesX, totalMatrixesY;

		if (modeButton != null) modeButton.setState(data.editMode ? 1 : 0);
		if (data.editMode)
setComment(I18n.message(config.getParameter("editModeComment"),
config.getParameter("MaxABSFillValue")));
		final int s = (auto.isAtStart() || auto.isAtEnd()) ? data.l : data.n;
		clientPane.removeAll();

		if (data.editMode) {
			calculateDimensions(s, width, height, 0.26, 0.23, 3, 3, 2.0, 1.0);
			addText(config.getParameter("editModeHeader"), width / 6, gapY / 2, 2 *
width / 3, 3 * gapY / 4, 3);
			drawMainMatrixes(0, gapY, width, height - gapY);
			return;
		}

		resetSelect();
		if ((data.rule == 0) && (!auto.isAtStart()) && (!auto.isAtEnd()))
			if (data.R == null) {
				data.A.updateRectBase(5);
				data.B.updateRectBase(5);
				data.A.getMatrixPart(1, 1, data.l, data.l).updateRectBase(0);
				data.B.getMatrixPart(1, 1, data.l, data.l).updateRectBase(0);
			} else {
				data.R.getMatrixPart(1, 1, data.l, data.l).updateRectBase(5);
				data.A.getMatrixPart(1, 1, data.l, data.l).updateRectBase(5);
				data.B.getMatrixPart(1, 1, data.l, data.l).updateRectBase(5);
			}
		if ((data.rule >= 1) && (data.rule <= 7) && (data.R == null))
			selectRuledMatrix(data.rule);

		if (auto.isAtStart() || auto.isAtEnd() || (data.rule == 0)) {
			calculateDimensions(s, width, height, 0.20, 0.64, 4, 2, 3.0, 1.0);
```

```
                    drawMainMatrixes(gapX, 0, sX + gapX + sX, height);
                    addText("=", 2 * (gapX + sX) + 1 + gapX / 4, gapY + (sY - sizeY) / 2,
gapX / 2, sizeY, 2);
                    if ((data.R == null) || (auto.isAtStart())) addText("?", 3 * gapX + 2 *
sizeX * s, gapY, sizeX * s, sizeY * s, 2);
                    else addMatrix(data.R, 3 * gapX + 2 * sizeX * s, gapY,
data.R.getMaxLength());
             } else {
                    calculateDimensions(s, width, height, 0.12, 0.20, 8, 6, 3.5, 1.5);
                    final int borderX = 3 * gapX + 2 * sX;
                    final int borderY = height - 2 * gapY - sY / 2;
                    if ((data.R != null) && (data.rule >= 7)) drawResultMatrix(borderX, 0,
width - borderX, borderY);
                    if ((data.rule >= 1) && (data.rule <= 7) && (data.R == null))
                           drawCurrentRule(borderX, 0, width - borderX, borderY);
                    drawDownPart(0, borderY, width, height - borderY);
                    drawMainMatrixes(gapX, 0, sX + gapX + sX, borderY);
             }
      }


      /**
       * Calculates global variables sizeX, sizeY, sX, sY, gapX and gapY
       * for using in drawing routines.
       *
       * @param s actual size of main matrixes.
       * @param width horizontal size of client pane.
       * @param height vertical size of client pane.
       * @param gapAspectX x-coefficient (gapX = sizeX * gapAspectX).
       * @param gapAspectY y-coefficient (gapY = sizeY * gapAspectY).
       * @param totalGapsX number of horizontal gaps between matrixes.
       * @param totalGapsY number of vertical gaps between matrixes.
       * @param totalMatrixesX number of full-size x-matrixes.
       * @param totalMatrixesY number of full-size y-matrixes.
       */
      private void calculateDimensions(int s, int width, int height,
                           double gapAspectX, double gapAspectY, int totalGapsX, int
totalGapsY,
                           double totalMatrixesX, double totalMatrixesY) {
             sizeX = (int)((width  / (totalGapsX * gapAspectX + totalMatrixesX)) / s);
             sizeY = (int)((height / (totalGapsY * gapAspectY + totalMatrixesY)) / s);
             sX = sizeX * s;
             sY = sizeY * s;
             gapX = (int)((width  - totalMatrixesX * sX) / totalGapsX);
             gapY = (int)((height - totalMatrixesY * sY) / totalGapsY);
      }


      /**
       * Put main matrixes to client pane.
       *
       * @param x1 left border for putting to client pane.
       * @param y1 up border for putting to client pane.
       * @param width horizontal size of this part of client pane.
       * @param height vertical size of this part of client pane.
       */
      private void drawMainMatrixes(int x1, int y1, int width, int height) {
             drawMatrixPair(data.A, data.B, x1 + width / 2 - gapX / 2 - sX, y1 + height / 2
- sY / 2);
      }
```

```
	/**
	 * Put one of seven calculating rules to client pane.
	 *
	 * @param x1 left border for putting to client pane.
	 * @param y1 up border for putting to client pane.
	 * @param width horizontal size of this part of client pane.
	 * @param height vertical size of this part of client pane.
	 */
	private void drawCurrentRule(int x1, int y1, int width, int height) {
		final int downSize = sY / 2 + gapY;
		final int textY = (height - downSize) / 7;
		final int ruleY = y1 + height - downSize + gapY / 4;
		final int ruleX = (x1 + width / 2) - (3 * sX / 2 + 2 * gapX) / 2;
		for (int i = 1; i <= 7; i++)
			addText("P" + i + " = " + rules[i], ruleX + 2 * gapX, y1 + textY * (i -
1), width - 2 * gapX, textY, 4);
		addText("*", ruleX + gapX / 2, y1 + textY * (data.rule - 1), gapX, textY, 3);

		Matrix pMatrix = new Matrix(data.P[data.rule]);
		addMatrix(pMatrix, ruleX, ruleY, pMatrix.getMaxLength());
		addText("=", ruleX + sX / 2 + 1 + gapX / 4, ruleY + (sY / 2 - sizeY) / 2, gapX
/ 2, sizeY, 2);
		drawMatrixPair(data.currentA, data.currentB, ruleX + sX / 2 + gapX, ruleY);
	}


	/**
	 * Put matrix with result and comments to client pane.
	 *
	 * @param x1 left border for putting to client pane.
	 * @param y1 up border for putting to client pane.
	 * @param width horizontal size of this part of client pane.
	 * @param height vertical size of this part of client pane.
	 */
	private void drawResultMatrix(int x1, int y1, int width, int height) {
		final int currentX = x1 + width  / 2 - sX / 2 - gapX / 4;
		final int currentY = y1 + height / 2 - sY / 2 - gapY / 4;
		final int textY = (sY + gapY) / 8;

		addText("R = P5 + P4 - P2 + P6", currentX, currentY - 2 * textY, sX + gapX /
2, textY, 4);
		addText("S = P1 + P2", currentX, currentY - textY, sX + gapX / 2, textY, 4);
		addText("T = P3 + P4", currentX, currentY + sY + gapY / 2, sX + gapX / 2,
textY, 4);
		addText("U = P5 + P1 - P3 + P7", currentX, currentY + sY + gapY / 2 + textY,
sX + gapX / 2, textY, 4);

		addCuttedMatrix(data.R, currentX, currentY, Math.max(Math.min(gapX / 2, gapY /
2), 1), data.R.getMaxLength());
		addText("R", currentX - gapX, currentY + (sY / 2 - gapY) / 2, gapX, gapY, 2);
		addText("S", currentX + sX + gapX / 2, currentY + (sY / 2 - gapY) / 2, gapX,
gapY, 2);
		addText("T", currentX - gapX, currentY + 3 * sY / 4 - gapY / 2, gapX, gapY,
2);
		addText("U", currentX + sX + gapX / 2, currentY + 3 * sY / 4 - gapY / 2, gapX,
gapY, 2);
	}
```

```
	/**
	 * Put state of Strassen matrix stack to client pane.
	 *
	 * @param x1 left border for putting to client pane.
	 * @param y1 up border for putting to client pane.
	 * @param width horizontal size of this part of client pane.
	 * @param height vertical size of this part of client pane.
	 */
	private void drawDownPart(int x1, int y1, int width, int height) {
		addText("", x1 + gapX / 2, y1, width - gapX - x1, 1, 3);
		int maxLength = 1;
		for (int i = 1; i <= data.rule; i++)
			if (data.P[i] != null) maxLength = Math.max(maxLength,
data.P[i].getMaxLength());
		int currentX = x1 + gapX;
		int qX = sizeX * data.A.getWidth() / 2;
		for (int i = 1; i <= data.rule; i++)
			if (data.P[i] != null) {
				addMatrix(data.P[i], currentX, y1 + gapY, maxLength);
				addText("P" + i, currentX, y1, qX, gapY, 2);
				currentX += (qX + gapX);
			}
	}



	/**
	 * Put state of two matrixes into client pane.
	 *
	 * @param A first matrix to put.
	 * @param B second matrix to put.
	 * @param x1 left border in client pane for putting matrixes.
	 * @param y1 up border in client pane for putting matrixes.
	 * @param maxLength number of ciphers for adjusting font size.
	 */
	private void drawMatrixPair(Matrix A, Matrix B, int x1, int y1, int maxLength) {
		int qX = sizeX * A.getWidth();
		int qY = sizeY * A.getHeight();
		int fontSize = (int)(qY / 8);
		int shift = fontSize / 6;
		addCuttedMatrix(A, x1, y1, 1, maxLength);
		addCuttedMatrix(B, x1 + gapX + qX, y1, 1, maxLength);
		if ((A.getWidth() == data.n) && (data.n > 1)) {
			addText("A", x1, y1 - fontSize + shift + 1, qX / 2, fontSize, 2);
			addText("B", x1 + qX / 2, y1 - fontSize + shift + 1, qX / 2, fontSize,
2);
			addText("C", x1, y1 + qY - shift + 1, qX / 2, fontSize, 2);
			addText("D", x1 + qX / 2, y1 + qY - shift + 1, qX / 2, fontSize, 2);
			addText("E", x1 + qX + gapX, y1 - fontSize + shift + 1, qX / 2,
fontSize, 2);
			addText("G", x1 + qX / 2 + qX + gapX, y1 - fontSize + shift + 1, qX /
2, fontSize, 2);
			addText("F", x1 + qX + gapX, y1 + qY - shift + 1, qX / 2, fontSize, 2);
			addText("H", x1 + qX / 2 + qX + gapX, y1 + qY - shift + 1, qX / 2,
fontSize, 2);
		}
		addText("X", x1 + qX + 1 + gapX / 4,y1 + (qY - sizeY) / 2, gapX / 2, sizeY,2);
	}
```

```
	/**
	 * Put state of two matrixes into client pane.
	 *
	 * @param A first matrix to put.
	 * @param B second matrix to put.
	 * @param x1 left border in client pane for putting matrixes.
	 * @param y1 up border in client pane for putting matrixes.
	 */
	private void drawMatrixPair(Matrix A, Matrix B, int x1, int y1) {
		drawMatrixPair(A, B, x1, y1, Math.max(A.getMaxLength(), B.getMaxLength()));
	}


	/**
	 * Put state of matrix splitted into four parts
	 * to client pane.
	 *
	 * @param A matrix to put.
	 * @param x1 left border in client pane for putting matrixes.
	 * @param y1 up border in client pane for putting matrixes.
	 * @param gap size of gap between matrix parts.
	 * @param maxLength number of ciphers for adjusting font size.
	 */
	private void addCuttedMatrix(Matrix A, int x1, int y1, int gap, int maxLength) {
		int size = A.getWidth();
		addMatrix(A, x1, y1, maxLength);
		if ((size == data.n) && (size > 1)) {
			int qX = sizeX * size / 2;
			int qY = sizeY * size / 2;
			addMatrix(A.getA(), x1, y1, maxLength);
			addMatrix(A.getB(), x1 + qX + gap, y1, maxLength);
			addMatrix(A.getC(), x1, y1 + qY + gap, maxLength);
			addMatrix(A.getD(), x1 + qX + gap, y1 + qY + gap, maxLength);
		}
	}


	/**
	 * Put state of matrix into client pane.
	 *
	 * @param A matrix to put.
	 * @param x1 left border in client pane for putting matrixes.
	 * @param y1 up border in client pane for putting matrixes.
	 * @param maxLength number of ciphers for adjusting font size.
	 */
	private void addMatrix(Matrix A, int x1, int y1, int maxLength) {
		String maxValueString = "99999999999999999999";
		int size = A.getWidth();
		A.createRectBase(styleSet);
		A.layoutMatrix(clientPane, x1, y1, x1 + sizeX * size, y1 + sizeY * size,
maxValueString.substring(0, maxLength));
	}
```

```
	/**
	 * Put any text into client pane.
	 *
	 * @param s text to put.
	 * @param x1 left border in client pane for putting text.
	 * @param y1 up border in client pane for putting text.
	 * @param width maximum possible width of text in client pane.
	 * @param height maximum possible height of text in client pane.
	 * @param styleNum number of style used for drawing.
	 */
	private void addText(String s, int x1, int y1, int width, int height, int styleNum) {
		Rect xRect = new Rect(styleSet, s, 2);
		xRect.setStyle(styleNum);
		xRect.setSize(width, height);
		xRect.setLocation(x1, y1);
		xRect.adjustFontSize();
		clientPane.add(xRect);
	}



	/**
	 * Select parts of matrix.
	 *
	 * @param rule number of rule which was used for calculating matrix.
	 */
	private void selectRuledMatrix(int rule) {
		StringTokenizer ruler = new StringTokenizer(rules[rule], " +-()x");
		while (ruler.hasMoreTokens())
			getNamedMatrix(ruler.nextToken().charAt(0), data.A,
data.B).updateRectBase(5);
	}


	/**
	 * Calculates part before 'x' in selected rule.
	 *
	 * @param rule number of rule to calculate for.
	 * @param A first matrix from which we can take parts.
	 * @param B second matrix from which we can take parts.
	 * @return matrix with result of expression before 'x'.
	 */
	public static Matrix getLeftResult(int rule, Matrix A, Matrix B) {
		StringTokenizer ruler = new StringTokenizer(rules[rule], "x");
		return calculateResult(ruler.nextToken(), A, B);
	}


	/**
	 * Calculates part after 'x' in selected rule.
	 *
	 * @param rule number of rule to calculate for.
	 * @param A first matrix from which we can take parts.
	 * @param B second matrix from which we can take parts.
	 * @return matrix with result of expression after 'x'.
	 */
	public static Matrix getRightResult(int rule, Matrix A, Matrix B) {
		StringTokenizer ruler = new StringTokenizer(rules[rule], "x");
		String leftPart = ruler.nextToken();
		return calculateResult(ruler.nextToken(), A, B);
	}
```

```
/**
 * Calculates expressions like '(A - B)', '(E + H)', 'G' etc.
 *
 * @param rule expressions to calculate.
 * @param A first matrix from which we can take parts.
 * @param B second matrix from which we can take parts.
 * @return matrix with result of expression.
 */
private static Matrix calculateResult(String rule, Matrix A, Matrix B) {
	StringTokenizer ruler = new StringTokenizer(rule, " ()");
	Matrix result = getNamedMatrix(ruler.nextToken().charAt(0), A, B);
	if (ruler.hasMoreTokens()) {
		char sign = ruler.nextToken().charAt(0);
		Matrix right = getNamedMatrix(ruler.nextToken().charAt(0), A, B);
		if (sign == '+') result = result.add(right);
		else result = result.sub(right);
	} else result = new Matrix(result);
	return result;
}


/**
 * Returns part of initial matrixes with choosed rule alias.
 *
 * @param c name of choosed matrix part (just rule alias of this part).
 * @param A first matrix from which we can take parts.
 * @param B second matrix from which we can take parts.
 * @return part of one of the initial matrixes.
 */
private static Matrix getNamedMatrix(char c, Matrix A, Matrix B) {
	if (c == 'A') return A.getA();
	if (c == 'B') return A.getB();
	if (c == 'C') return A.getC();
	if (c == 'D') return A.getD();
	if (c == 'E') return B.getA();
	if (c == 'F') return B.getC();
	if (c == 'G') return B.getB();
			return B.getD();
}




/**
 * Reset possible selection of elements for matrix pair.
 */
private void resetSelect() {
	if (data.A != null) data.A.updateRectBase(0);
	if (data.B != null) data.B.updateRectBase(0);
	if (data.R != null) data.R.updateRectBase(0);
}


/**
 * Deselects active cell in edit mode.
 */
private void resetActiveCell() {
	handleActiveCell = "";
	negativeActiveCell = false;
	if (activeCell != null) {
		activeCell.updateRectBase(0);
		activeCell = null;
	}
}
```

```
	/**
	 * Handler of mouse clicks in edit mode.
	 *
	 * @param e mouse event to handle.
	 */
	public void mouseClicked(MouseEvent e) {
		if (data.editMode) {
			Cell previousCell = activeCell;
			resetActiveCell();
			int x = e.getX();
			int y = e.getY();
			activeCell = data.A.inside(x, y);
			if (activeCell == null)   activeCell = data.B.inside(x, y);
			if (activeCell != null) {
				if (activeCell == previousCell) activeCell = null;
				else activeCell.updateRectBase(1);
			}
			updateScreen();
		}
	}


	/**
	 * Keyboard handler a in edit mode.
	 *
	 * @param e key event to handle.
	 */
	public void keyTyped(KeyEvent e) {
		char c = e.getKeyChar();
		int code = (int) c;
		if ((data.editMode) && (activeCell != null) && ((code == 8) || (code == 10) ||
(code == 27) || Character.isDigit(c) || (c == '-'))) {
			if ((code == 10) || (code == 27)) resetActiveCell();
			else {                          // Change value in active cell
				if (c == '-') negativeActiveCell = !negativeActiveCell;
				String result = handleActiveCell;
				if (Character.isDigit(c)) result = result + c;
				int l = result.length();
				if (code == 8) {
					if (l > 0) result = result.substring(0, l - 1);
					else negativeActiveCell = false;
				}
				int resultI = 0;
				if (result.length() > 0)
					resultI = Integer.parseInt(result);
				if (resultI <= config.getInteger("MaxABSFillValue")) {
					handleActiveCell = result;
					if (negativeActiveCell) resultI = -resultI;
					activeCell.setValue(resultI);
				}
			}
			updateScreen();
		}
	}

    /**
     * Invoked on adjustment event.
     *
     * @param event event to process.
     */
    public void adjustmentValueChanged(AdjustmentEvent event) {
        if (event.getSource() == sizePanel) {
			data.l = event.getValue();
			data.n = 1;
			while (data.n < data.l) data.n *= 2;
```

```
                    setRandomMatrixes();
                }
            }

                    // Just not used methods inherited from
                    // MouseListener and KeyListener
              public void mouseExited(MouseEvent e) { }
              public void mouseEntered(MouseEvent e) { }
              public void mousePressed(MouseEvent e) { }
              public void mouseReleased(MouseEvent e) { }
              public void keyPressed(KeyEvent e) { }
              public void keyReleased(KeyEvent e) { }
        }
```

Приложение 2. *XML*-описание визуализатора

Файл Strassen.xml (основные параметры визуализатора)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "Strassen" visualizer description
    Version: $Id: Strassen.xml,v 1.3 2004/01/12 23:55:04 Kot Exp $
-->

<!DOCTYPE visualizer PUBLIC
    "-//IFMO Vizi//Visualizer description"
    "http://ips.ifmo.ru/vizi/dtd/visualizer.dtd"
[
    <!ENTITY algorithm SYSTEM "Strassen-Algorithm.xml">
    <!ENTITY configuration SYSTEM "Strassen-Configuration.xml">
]>

<visualizer
    id="Strassen"
    package="ru.ifmo.vizi.strassen"
    main-class="StrassenVisualizer"

    preferred-width="585"
    preferred-height="460"

    name-ru="Алгоритм Штрассена"
    name-en="Strassen algorythm"

    author-ru="Александр Котов"
    author-en="Alex Kotov"
    author-email="kotovs@yahoo.com"

    supervisor-ru="Георгий Корнеев"
    supervisor-en="Georgiy Korneev"
    supervisor-email="kgeorgiy@rain.ifmo.ru"

    copyright-ru="Copyright \u00A9 Кафедра КТ, СПб ГИТМО (ТУ), 2004"
    copyright-en="Copyright \u00A9 Computer Technologies Department, SPb IFMO, 2004"
>
    &algorithm;
    &configuration;
</visualizer>
```

Файл Strassen-Algorithm.xml (xml-представление алгоритма)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "Strassen" algoritm description
    Version: $Id: Strassen-Algorithm.xml,v 0.9 2004/01/15 02:20:04 Kot Exp $
-->

<algorithm>
      <variable
            description="Максимальный размер матриц"
            name="maxSize"
            type="int"
            value="8"
      />
```

```
<variable
	description="Размер подготовленных матриц"
	name="n"
	type="int"
	value="4"
/>
<variable
	description="Исходный размер матриц"
	name="l"
	type="int"
	value="4"
/>
<variable
	description="Номер текущего умножения (1..7)"
	name="rule"
	type="int"
	value="0"
/>
<variable
	description="Первая исходная умножаемая матрица"
	name="A"
	type="Matrix"
	value="new Matrix(4, 4)"
/>
<variable
	description="Вторая исходная умножаемая матрица"
	name="B"
	type="Matrix"
	value="new Matrix(4, 4)"
/>
<variable
	description="Первая промежуточная умножаемая матрица"
	name="currentA"
	type="Matrix"
	value="null"
/>
<variable
	description="Вторая промежуточная умножаемая матрица"
	name="currentB"
	type="Matrix"
	value="null"
/>
<variable
	description="Матрица результата умножения"
	name="R"
	type="Matrix"
	value="null"
/>
<variable
	description="Семь промежуточных матриц"
	name="P"
	type="Matrix[]"
	value="new Matrix[8]"
/>
<variable
	description="Экземпляр апплета"
	name="applet"
	type="StrassenVisualizer"
	value="null"
/>
<variable
	description="Текущий режим работы апплета"
	name="editMode"
	type="boolean"
	value="false"
```

```
  />
  <variable
        description="Текущий шаг алгоритма"
        name="step"
        type="int"
        value="0"
  />


  <toString>
        StringBuffer s = new StringBuffer();
        s.append("size=" + l + " editMode=" + editMode);
        return s.toString();
  </toString>



<auto id="Main" description="Алгоритм Штрассена умножения матриц">

  <start
        comment-ru="Две матрицы размера {0}x{0} перед умножением рекурсивным методом
        Штрассена"
        comment-en="Two matrix {0}x{0} size before recursive Strassen matrix
        multiplication"
        comment-args="new Integer(@l)"
  >
        <draw>@applet.updateScreen();</draw>
  </start>

  <step
        description="Подготовка к умножению матриц"
        level="-1"
  >
        <direct>
              @step = 0;
              @rule = 0;
              @R = null;
              @editMode = false;
        </direct>
  </step>

  <if description="Размер умножаемых матриц не является степенью 2?"
        level="-1"
        test="@l != @n"
  ><then>
        <step
              id="StartMatrixResize"
              description="Увеличение размера матриц до степени 2"
              comment-ru="Размер исходных матриц {0}x{0} не является натуральной
              степенью двойки, поэтому дополняем исходные матрицы до размера
              {1}x{1}."
              comment-en="We do not have size of our source matrixes {0}x{0} equals
              to natural power of 2, so we enlarge size of matrixes up to {1}x{1}."
              comment-args="new Integer(@l), new Integer(@n)"
              level="0"
        >
              <draw>@applet.updateScreen();</draw>
              <action>
                    @step @= @step + 1;
                    @A @= @A.getMatrixPart(1, 1, @n, @n);
                    @B @= @B.getMatrixPart(1, 1, @n, @n);
              </action>
        </step>
  </then></if>
```

```
<if description="Нужно ли нам использовать метод Штрассена?"
      level="-1"
      test="@n > 1"
><then>
      <while description="Цикл семи Штрассеновских умножений"
            level="-1"
            test="@rule &lt; 7"
      >
            <step
                  id="OneOfSevenMultiplications"
                  description="Одно из семи умножений"
                  comment-ru="Проводим вспомогательное умножение №{1} матриц
                  размера {0}x{0}.."
                  comment-en="Matrix {0}x{0} size multiplication No{1}.."
                  comment-args="new Integer(@n / 2), new Integer(@rule)"
                  level="0"
            >
                  <draw>@applet.updateScreen();</draw>
                  <action>
                        @step @= @step + 1;
                        @rule @= @rule + 1;
                        @currentA @= StrassenVisualizer.getLeftResult(@rule, @A,
                        @B);
                        @currentB @= StrassenVisualizer.getRightResult(@rule, @A,
                        @B);
                        @P[@rule] @= @currentA.mul(@currentB);
                  </action>
            </step>
      </while>

      <step
            id="MultiplicationResult"
            description="Получение матрицы результата"
            comment-ru="Из полученных произведений в серии семи умножений матриц
            размера {0}x{0}, формируется матрицу результата."
            comment-en="Constructing matrix with result from series of seven matrix
            size {0}x{0} multiplications."
            comment-args="new Integer(@n / 2)"
            level="0"
      >
            <draw>@applet.updateScreen();</draw>
            <action>
                  @step @= @step + 1;
                  Matrix r = @P[5].add(@P[4]).sub(@P[2]).add(@P[6]);
                  Matrix s = @P[1].add(@P[2]);
                  Matrix t = @P[3].add(@P[4]);
                  Matrix u = @P[5].add(@P[1]).sub(@P[3]).sub(@P[7]);
                  @R @= new Matrix(r, s, t, u);
            </action>
      </step>
</then><else>
      <step
            description="Простое умножение исходных матриц"
            level="-1"
      >
            <action>@R @= @A.mul(@B);</action>
      </step>
</else></if>



<if description="Размер исходных матриц не является степенью 2?"
      level="-1"
      test="@l != @n"
><then>
```

```
            <step
                  id="FinishMatrixResize"
                  description="Намерение уменьшить размер матриц до исходного"
                  comment-ru="Мы получили произведение матриц размера {1}x{1}. Поскольку
                  исходный размер матриц не является натуральной степенью двойки, усекаем
                  матрицу результата до исходного размера {0}x{0}."
                  comment-en="We have received matrix product size {1}x{1}. Just because
                  initial matrix size not equals to natural power of 2, we cut result
                  matrix to initial size {0}x{0}."
                  comment-args="new Integer(@l), new Integer(@n)"
                  level="0"
            >
                  <draw>@applet.updateScreen();</draw>
                  <action>
                        @rule @= 0;
                        @step @= @step + 1;
                  </action>
            </step>
            <step
                  description="Уменьшение размера матриц до исходного"
                  level="-1"
            >
                  <action>
                        @R @= @R.getMatrixPart(1, 1, @l, @l);
                        @A @= @A.getMatrixPart(1, 1, @l, @l);
                        @B @= @B.getMatrixPart(1, 1, @l, @l);
                  </action>
            </step>
      </then></if>

      <step
            description="Корректное завершение алгоритма"
            level="-1"
      >
            <action>@step @= @step + 1;</action>
      </step>

      <finish
            comment-ru="Алгоритм Штрассена умножения матриц завершил свою работу."
            comment-en="Strassen matrix multiplication algorythm finished its work."
      >
            <draw>@applet.updateScreen();</draw>
      </finish>

      </auto>
</algorithm>
```

Файл Strassen-Configuration.xml (конфигурация визуализатора)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>

<!--
    "Strassen" visualizer configuration (example).
    Shared for Strassen.xml
    Version: $Id: Strassen-Configuration.xml,v 1.2 2004/01/12 11:42:21 Kot Exp $
-->

<configuration>
      <property
            description = "Maximum possible matrix size in applet"
            param       = "maxMatrixSize"
            value       = "8"
```

```
/>
<property
      description = "Matrix size when applet started"
      param       = "defaultMatrixSize"
      value       = "4"
/>
<property
      description = "Maximum absolute value for filling matrixes"
      param       = "MaxABSFillValue"
      value       = "20"
/>
<message
      description = "Caption for edit mode"
      param     = "editModeHeader"
      message-ru  = "Режим редактирования умножаемых матриц"
      message-en  = "Edit mode for matrixes"
/>
<message
      description = "Comment in client pane for edit mode"
      param     = "editModeComment"
      message-ru  = "Щелкните на ячейке матрицы, которую хотите изменить и введите
      целое число от -{0} до {0}."
      message-en  = "Click on cell of the matrix which you want to change, then
      enter integer number in range -{0}..{0}."
/>
                  <!-- APPEARANCE -->
<styleset
      description = "Matrix style set"
      param       = "matrix-style"
>
      <style
            description     = "Default style"
            text-color      = "000000"
            text-align      = "0.5"
            border-color    = "000000"
            border-status   = "true"
            fill-color      = "ffffc0"
            fill-status     = "true"
            aspect-status   = "false"
            padding         = "0.2"
      >
            <font
                  face        = "Serif"
                  size        = "12"
                  style       = "plain"
            />
      </style>
      <style
            description     = "Selected style"
            fill-color      = "c0e0c0"
      />
      <style
            description     = "Simple text output style"
            fill-status     = "false"
            border-status   = "false"
      />
      <style
            description     = "Bordered text output style"
            fill-color      = "f0f0f0"
            border-color    = "c0c0c0"
      >
            <font
                  face        = "Monospaced"
                  size        = "12"
                  style       = "plain"
            />
```

```
            </style>
            <style
                  description        = "Left-aligned text style"
                  fill-status        = "false"
                  border-status= "false"
                  message-align= "0"
            />
            <style
                  description     = "Represented style"
                  fill-color      = "ffe0c0"
            />
      </styleset>


<!-- Appearance properties -->
      <property
            description = "Comment pane height"
            param       = "comment-height"
            value       = "40"
      />


<!-- controlsPane configuration -->
    <adjustablePanel
            description = "'Size' button"
            param       = "size"
            caption-ru  = "Матрицы {0,number,####}x{0,number,####}"
            caption-en  = "Matrixes {0,number,####}x{0,number,####}"
            hint-ru     = "Установить размер матриц"
            hint-en     = "Set matrix size"
            value       = "1"
        minimum         = "1"
        maximum         = "64"
        unitIncrement   = "1"
        blockIncrement  = "100"
        blockInterval   = "100"
    >
            <button
                  description = "Button for decreasing matrix size"
                  param       = "decrementButton"
                  caption-ru  = "&lt;&lt;"
                  caption-en  = "&lt;&lt;"
                  hint-ru     = "Уменьшить размер матриц"
                  hint-en      = "Decrease matrix size"
            />
            <button
                  description = "Button for increasing matrix size"
                  param       = "incrementButton"
                  caption-ru  = "&gt;&gt;"
                  caption-en  = "&gt;&gt;"
                  hint-ru     = "Увеличить размер матриц"
                  hint-en     = "Increase matrix size"
            />
    </adjustablePanel>

      <message
            description = "Caption of join button in controlsPane"
            param     = "button-random"
            message-ru  = "Случайно"
            message-en  = "Randomize"
      />
      <message
            description = "Hint for join button in controlsPane"
            param     = "button-random-hint"
            message-ru  = "Заполнить умножаемые матрицы случайными числами"
            message-en  = "Fill matrixes with random numbers"
```

```
	/>
	<message
		description = "Caption of join button in controlsPane"
		param     = "button-edit"
		message-ru  = "Редактировать"
		message-en  = "Edit mode"
	/>
	<message
		description = "Hint for join button in controlsPane"
		param     = "button-edit-hint"
		message-ru  = "Переключиться в режим редактирования"
		message-en  = "Switch to matrix edit mode"
	/>
	<message
		description = "Caption of join button in controlsPane"
		param     = "button-show"
		message-ru  = "Режим показа"
		message-en  = "Show mode"
	/>
	<message
		description = "Hint for join button in controlsPane"
		param     = "button-show-hint"
		message-ru  = "Переключиться в режим умножения матриц"
		message-en  = "Switch to matrix multiplication show mode"
	/>


<!-- load/save window configuration -->
	<group
		description = "Save/Load dialog configuration"
		param       = "SaveLoadDialog"
	>
		<property
			description = "Height of the comment pane"
			param       = "CommentPane-lines"
			value       = "1"
		/>
		<property
			description = "Width of the text area"
			param       = "columns"
			value       = "60"
		/>
		<property
			description = "Height of the text area"
			param       = "rows"
			value       = "5"
		/>
	</group>

	<message
		description = "Hint string for save/load window"
		param     = "loadStateHint"
		message-ru  = "Числа, задающие матрицу - ее элементы в порядке слева-направо
		сверху-вниз."
		message-en  = "Matrix definition numbers - it\'s elements in left-to-right up-
		to-down order."
	/>
	<message
		description = "Very big matrix error in save/load window"
		param     = "incorrectState1"
		message-ru  = "Ошибка: размер матриц слишком большой."
		message-en  = "Error: matrix size is too big."
	/>
</configuration>
```

Приложение 3. Сгенерированные исходные коды автомата.

```
package ru.ifmo.vizi.strassen;

import ru.ifmo.vizi.base.auto.*;
import java.util.Locale;

public final class Strassen extends BaseAutoReverseAutomata {
    /**
      * Модель данных.
      */
    public final Data d = new Data();

    /**
      * Конструктор для языка
      */
    public Strassen(Locale locale) {
        super("ru.ifmo.vizi.strassen.Comments", locale);
        init(new Main(), d);
    }

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Максимальный размер матриц.
          */
        public int maxSize = 8;

        /**
          * Размер подготовленных матриц.
          */
        public int n = 4;

        /**
          * Исходный размер матриц.
          */
        public int l = 4;

        /**
          * Номер текущего умножения (1..7).
          */
        public int rule = 0;
```

```
/**
  * Первая исходная умножаемая матрица.
  */
public Matrix A = new Matrix(4, 4);

/**
  * Вторая исходная умножаемая матрица.
  */
public Matrix B = new Matrix(4, 4);

/**
  * Первая промежуточная умножаемая матрица.
  */
public Matrix currentA = null;

/**
  * Вторая промежуточная умножаемая матрица.
  */
public Matrix currentB = null;

/**
  * Матрица результата умножения.
  */
public Matrix R = null;

/**
  * Семь промежуточных матриц.
  */
public Matrix[] P = new Matrix[8];

/**
  * Экземпляр апплета.
  */
public StrassenVisualizer applet = null;

/**
  * Текущий режим работы апплета.
  */
public boolean editMode = false;

/**
  * Текущий шаг алгоритма.
  */
```

```
    public int step = 0;

    public String toString() {
               StringBuffer s = new StringBuffer();
               s.append("size=" + l + " editMode=" + editMode);
               return s.toString();
    }
}

/**
  * Алгоритм Штрассена умножения матриц.
  */
private final class Main extends BaseAutomata implements Automata {
    /**
      * Начальное состояние автомата.
      */
    private final int START_STATE = 0;

    /**
      * Конечное состояние автомата.
      */
    private final int END_STATE = 16;

    /**
      * Конструктор.
      */
    public Main() {
        super(
            "Main",
            0, // Номер начального состояния
            16, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Подготовка к умножению матриц",
                "Размер умножаемых матриц не является степенью 2?",
                "Размер умножаемых матриц не является степенью 2? (окончание)",
                "Увеличение размера матриц до степени 2",
                "Нужно ли нам использовать метод Штрассена?",
                "Нужно ли нам использовать метод Штрассена? (окончание)",
                "Цикл семи Штрассеновских умножений",
                "Одно из семи умножений",
                "Получение матрицы результата",
                "Простое умножение исходных матриц",
                "Размер исходных матриц не является степенью 2?",
```

```
            "Размер исходных матриц не является степенью 2? (окончание)",
            "Намерение уменьшить размер матриц до исходного",
            "Уменьшение размера матриц до исходного",
            "Корректное завершение алгоритма",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            -1, // Подготовка к умножению матриц
            -1, // Размер умножаемых матриц не является степенью 2?
            -1, // Размер умножаемых матриц не является степенью 2? (окончание)
            0, // Увеличение размера матриц до степени 2
            -1, // Нужно ли нам использовать метод Штрассена?
            -1, // Нужно ли нам использовать метод Штрассена? (окончание)
            -1, // Цикл семи Штрассеновских умножений
            0, // Одно из семи умножений
            0, // Получение матрицы результата
            -1, // Простое умножение исходных матриц
            -1, // Размер исходных матриц не является степенью 2?
            -1, // Размер исходных матриц не является степенью 2? (окончание)
            0, // Намерение уменьшить размер матриц до исходного
            -1, // Уменьшение размера матриц до исходного
            -1, // Корректное завершение алгоритма
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата в перед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Подготовка к умножению матриц
            break;
        }
        case 1: { // Подготовка к умножению матриц
            state = 2; // Размер умножаемых матриц не является степенью 2?
            break;
        }
        case 2: { // Размер умножаемых матриц не является степенью 2?
            if (d.l != d.n) {
                state = 4; // Увеличение размера матриц до степени 2
```

```
    } else {
        stack.pushBoolean(false);
       state = 3; // Размер умножаемых матриц не является степенью 2?
          (окончание)
    }
    break;
}
case 3: { // Размер умножаемых матриц не является степенью 2? (окончание)
    state = 5; // Нужно ли нам использовать метод Штрассена?
    break;
}
case 4: { // Увеличение размера матриц до степени 2
    stack.pushBoolean(true);
    state = 3; // Размер умножаемых матриц не является степенью 2?
          (окончание)
    break;
}
case 5: { // Нужно ли нам использовать метод Штрассена?
    if (d.n > 1) {
        stack.pushBoolean(false);
        state = 7; // Цикл семи Штрассеновских умножений
    } else {
        state = 10; // Простое умножение исходных матриц
    }
    break;
}
case 6: { // Нужно ли нам использовать метод Штрассена? (окончание)
    state = 11; // Размер исходных матриц не является степенью 2?
    break;
}
case 7: { // Цикл семи Штрассеновских умножений
    if (d.rule < 7) {
        state = 8; // Одно из семи умножений
    } else {
        state = 9; // Получение матрицы результата
    }
    break;
}
case 8: { // Одно из семи умножений
    stack.pushBoolean(true);
    state = 7; // Цикл семи Штрассеновских умножений
    break;
}
case 9: { // Получение матрицы результата
```

```
            stack.pushBoolean(true);
            state = 6; // Нужно ли нам использовать метод Штрассена? (окончание)
            break;
        }
        case 10: { // Простое умножение исходных матриц
            stack.pushBoolean(false);
            state = 6; // Нужно ли нам использовать метод Штрассена? (окончание)
            break;
        }
        case 11: { // Размер исходных матриц не является степенью 2?
            if (d.l != d.n) {
                state = 13; // Намерение уменьшить размер матриц до исходного
            } else {
                stack.pushBoolean(false);
                state = 12; // Размер исходных матриц не является степенью 2?
                  (окончание)
            }
            break;
        }
        case 12: { // Размер исходных матриц не является степенью 2? (окончание)
            state = 15; // Корректное завершение алгоритма
            break;
        }
        case 13: { // Намерение уменьшить размер матриц до исходного
            state = 14; // Уменьшение размера матриц до исходного
            break;
        }
        case 14: { // Уменьшение размера матриц до исходного
            stack.pushBoolean(true);
            state = 12; // Размер исходных матриц не является степенью 2?
             (окончание)
            break;
        }
        case 15: { // Корректное завершение алгоритма
            state = END_STATE;
            break;
        }
    }

    // Действие в текущем состоянии
    switch (state) {
        case 1: { // Подготовка к умножению матриц
                                d.step = 0;
                                d.rule = 0;
```

```
                    d.R = null;
                    d.editMode = false;
    break;
}
case 2: { // Размер умножаемых матриц не является степенью 2?
    break;
}
case 3: { // Размер умножаемых матриц не является степенью 2? (окончание)
    break;
}
case 4: { // Увеличение размера матриц до степени 2
    startSection();
    storeField(d, "step");
                            d.step = d.step + 1;
    storeField(d, "A");
                            d.A = d.A.getMatrixPart(1, 1, d.n, d.n);
    storeField(d, "B");
                            d.B = d.B.getMatrixPart(1, 1, d.n, d.n);

    break;
}
case 5: { // Нужно ли нам использовать метод Штрассена?
    break;
}
case 6: { // Нужно ли нам использовать метод Штрассена? (окончание)
    break;
}
case 7: { // Цикл семи Штрассеновских умножений
    break;
}
case 8: { // Одно из семи умножений
    startSection();
    storeField(d, "step");
   d.step = d.step + 1;
    storeField(d, "rule");
   d.rule = d.rule + 1;
    storeField(d, "currentA");
   d.currentA = StrassenVisualizer.getLeftResult(d.rule, d.A, d.B);
    storeField(d, "currentB");
   d.currentB = StrassenVisualizer.getRightResult(d.rule, d.A, d.B);
    storeArray(d.P, d.rule);
   d.P[d.rule] = d.currentA.mul(d.currentB);
    break;
}
```

```
case 9: { // Получение матрицы результата
    startSection();
    storeField(d, "step");
   d.step = d.step + 1;
   Matrix r = d.P[5].add(d.P[4]).sub(d.P[2]).add(d.P[6]);
   Matrix s = d.P[1].add(d.P[2]);
   Matrix t = d.P[3].add(d.P[4]);
   Matrix u = d.P[5].add(d.P[1]).sub(d.P[3]).sub(d.P[7]);
    storeField(d, "R");
   d.R = new Matrix(r, s, t, u);
    break;
}
case 10: { // Простое умножение исходных матриц
    startSection();
    storeField(d, "R");
    d.R = d.A.mul(d.B);
    break;
}
case 11: { // Размер исходных матриц не является степенью 2?
    break;
}
case 12: { // Размер исходных матриц не является степенью 2? (окончание)
    break;
}
case 13: { // Намерение уменьшить размер матриц до исходного
    startSection();
    storeField(d, "rule");
   d.rule = 0;
    storeField(d, "step");
   d.step = d.step + 1;
    break;
}
case 14: { // Уменьшение размера матриц до исходного
    startSection();
    storeField(d, "R");
   d.R = d.R.getMatrixPart(1, 1, d.l, d.l);
    storeField(d, "A");
   d.A = d.A.getMatrixPart(1, 1, d.l, d.l);
    storeField(d, "B");
   d.B = d.B.getMatrixPart(1, 1, d.l, d.l);
    break;
}
case 15: { // Корректное завершение алгоритма
    startSection();
```

```
            storeField(d, "step");
            d.step = d.step + 1;
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Подготовка к умножению матриц
            break;
        }
        case 2: { // Размер умножаемых матриц не является степенью 2?
            break;
        }
        case 3: { // Размер умножаемых матриц не является степенью 2? (окончание)
            break;
        }
        case 4: { // Увеличение размера матриц до степени 2
            restoreSection();
            break;
        }
        case 5: { // Нужно ли нам использовать метод Штрассена?
            break;
        }
        case 6: { // Нужно ли нам использовать метод Штрассена? (окончание)
            break;
        }
        case 7: { // Цикл семи Штрассеновских умножений
            break;
        }
        case 8: { // Одно из семи умножений
            restoreSection();
            break;
        }
        case 9: { // Получение матрицы результата
            restoreSection();
            break;
        }
        case 10: { // Простое умножение исходных матриц
```

```
            restoreSection();
            break;
        }
        case 11: { // Размер исходных матриц не является степенью 2?
            break;
        }
        case 12: { // Размер исходных матриц не является степенью 2? (окончание)
            break;
        }
        case 13: { // Намерение уменьшить размер матриц до исходного
            restoreSection();
            break;
        }
        case 14: { // Уменьшение размера матриц до исходного
            restoreSection();
            break;
        }
        case 15: { // Корректное завершение алгоритма
            restoreSection();
            break;
        }
    }

    // Переход в предыдущее состояние
    switch (state) {
        case 1: { // Подготовка к умножению матриц
            state = START_STATE;
            break;
        }
        case 2: { // Размер умножаемых матриц не является степенью 2?
            state = 1; // Подготовка к умножению матриц
            break;
        }
        case 3: { // Размер умножаемых матриц не является степенью 2? (окончание)
            if (stack.popBoolean()) {
                state = 4; // Увеличение размера матриц до степени 2
            } else {
                state = 2; // Размер умножаемых матриц не является степенью 2?
            }
            break;
        }
        case 4: { // Увеличение размера матриц до степени 2
            state = 2; // Размер умножаемых матриц не является степенью 2?
            break;
```

```
}
case 5: { // Нужно ли нам использовать метод Штрассена?
    state = 3; // Размер умножаемых матриц не является степенью 2?
          (окончание)
    break;
}
case 6: { // Нужно ли нам использовать метод Штрассена? (окончание)
    if (stack.popBoolean()) {
        state = 9; // Получение матрицы результата
    } else {
        state = 10; // Простое умножение исходных матриц
    }
    break;
}
case 7: { // Цикл семи Штрассеновских умножений
    if (stack.popBoolean()) {
        state = 8; // Одно из семи умножений
    } else {
        state = 5; // Нужно ли нам использовать метод Штрассена?
    }
    break;
}
case 8: { // Одно из семи умножений
    state = 7; // Цикл семи Штрассеновских умножений
    break;
}
case 9: { // Получение матрицы результата
    state = 7; // Цикл семи Штрассеновских умножений
    break;
}
case 10: { // Простое умножение исходных матриц
    state = 5; // Нужно ли нам использовать метод Штрассена?
    break;
}
case 11: { // Размер исходных матриц не является степенью 2?
    state = 6; // Нужно ли нам использовать метод Штрассена? (окончание)
    break;
}
case 12: { // Размер исходных матриц не является степенью 2? (окончание)
    if (stack.popBoolean()) {
        state = 14; // Уменьшение размера матриц до исходного
    } else {
        state = 11; // Размер исходных матриц не является степенью 2?
    }
```

```
            break;
        }
        case 13: { // Намерение уменьшить размер матриц до исходного
            state = 11; // Размер исходных матриц не является степенью 2?
            break;
        }
        case 14: { // Уменьшение размера матриц до исходного
            state = 13; // Намерение уменьшить размер матриц до исходного
            break;
        }
        case 15: { // Корректное завершение алгоритма
           state = 12; // Размер исходных матриц не является степенью 2?
                 (окончание)
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 15; // Корректное завершение алгоритма
            break;
        }
    }
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case START_STATE: { // Начальное состояние
            comment = Strassen.this.getComment("Main.START_STATE");
            args = new Object[]{new Integer(d.l)};
            break;
        }
        case 4: { // Увеличение размера матриц до степени 2
            comment = Strassen.this.getComment("Main.StartMatrixResize");
            args = new Object[]{new Integer(d.l), new Integer(d.n)};
            break;
        }
        case 8: { // Одно из семи умножений
            comment = Strassen.this.getComment("Main.OneOfSevenMultiplications");
            args = new Object[]{new Integer(d.n / 2), new Integer(d.rule)};
            break;
```

```
        }
        case 9: { // Получение матрицы результата
            comment = Strassen.this.getComment("Main.MultiplicationResult");
            args = new Object[]{new Integer(d.n / 2)};
            break;
        }
        case 13: { // Намерение уменьшить размер матриц до исходного
            comment = Strassen.this.getComment("Main.FinishMatrixResize");
            args = new Object[]{new Integer(d.l), new Integer(d.n)};
            break;
        }
        case END_STATE: { // Конечное состояние
            comment = Strassen.this.getComment("Main.END_STATE");
            break;
        }
    }

    return java.text.MessageFormat.format(comment, args);
}

/**
  * Выполняет действия по отрисовке состояния
  */
public void drawState() {
    switch (state) {
        case START_STATE: { // Начальное состояние
            d.applet.updateScreen();
            break;
        }
        case 4: { // Увеличение размера матриц до степени 2
            d.applet.updateScreen();
            break;
        }
        case 8: { // Одно из семи умножений
            d.applet.updateScreen();
            break;
        }
        case 9: { // Получение матрицы результата
            d.applet.updateScreen();
            break;
        }
        case 13: { // Намерение уменьшить размер матриц до исходного
            d.applet.updateScreen();
            break;
```

```
                }
                case END_STATE: { // Конечное состояние
                    d.applet.updateScreen();
                    break;
                }
            }
        }
    }
}
```