Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А. И. Пономарев

Построение визуализатора алгоритма поразрядной сортировки набора целых чисел 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 была предложена технология *Vizi* [1]. Визуализатор – это программа, позволяющая удобно и наглядно изучать работу некого алгоритма, демонстрируя его выполнение на неком наборе данных.

В настоящей работе применение технологии *Vizi* проиллюстрировано на примере разработки визуализатора алгоритма поразрядной сортировки набора целых чисел [2].

1. Описание алгоритма

Основной особенностью данного алгоритма является то, что вместо непосредственного сравнения элементов, производится разделение значения на части — *разряды* ключа. До сортировки необходимо знать два параметра: k и m, где k — разрядность самого длинного ключа, а m — количество возможных значений разряда ключа. В представленном визуализаторе количество разрядов ограничено четырьмя, а m равно десяти, поскольку рассматриваются десятичные числа.

Предположим, что элементы массива A есть k-разрядные десятичные числа, разрядность максимального числа известна заранее. Обозначим $d(j,n)$ — j-ю справа цифру числа n. Пусть B вспомогательный массив (в начале пустой) того же размера что и A, а *counter* — массив счетчиков из m элементов, заполненный нулями. Для каждого $j = 1, 2,..., k$ необходимо:

1. Пройти по массиву A, увеличивая на единицу значение счетчика с индексом $d(j,n)$.
2. Пройти по массиву *counter*, записывая в каждую ячейку сумму всех предыдущих элементов этого массива. Эта сумма является количеством чисел исходного массива, которые после сортировки должны будут находиться слева от числа, содержащего в j-ом разряде цифру, равную индексу этого счетчика. Таким образом, получается позиция этого числа в результирующем массиве.
3. Скопировать в массив B значения элементов исходного массива. При этом значение

n-го элемента массива *A* переносится в ячейку с индексом *counter*[*d*(*j*,*n*)]. Поскольку значения *j*-го разряда элементов массива *A* могут повторяться, то после копирования каждого элемента следует увеличить на единицу значение соответствующего счетчика. После заполнения массива *B* все элементы из него копируются в массив *A*. Затем происходит переход к следующему значению переменной *j*.

2. Исходный текст программы

Исходный текст программы на языке *Java*, реализующей рассматриваемый алгоритм, приведен в Приложении 1.

3. Описание модели данных

Моделью данных называется класс, содержащий все, необходимые алгоритму, переменные и структуры данных. Для реализации алгоритма поразрядной сортировки набора целых чисел требуется модель данных, содержащая:

- сортируемый массив целых чисел *a*;
- второй массив для временного хранения данных *res*;
- массив счетчиков *counter*;
- переменная цикла по разрядам *i*;
- переменная цикла по элементам массива *j*;
- переменная для хранения текущей цифры *digit*;
- переменная для хранения суммы значения счетчиков *v*;
- текущая степень 10 *p*;
- экземпляр апплета *visualizer*.

4. Преобразование программы

В Приложении 2 приведен текст преобразованной программы, 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*).

5. Описание преобразованной программы на языке *XML*

Приложение 3 содержит описание преобразованной программы на языке *XML.* Это приложение содержит три файла.

Первый файл *RadixSort.xml* содержит тег *visualizer*, хранящий общую информацию, такую как название визуализатора, имя автора и т. д., а также ссылки на два других файла — *RadixSort-Algorithm.xml* и *RadixSort-Configuration.xml.*

Второй файл *RadixSort-Algorithm.xml* содержит *XML* – описание модели данных и алгоритма. Описание модели данных размещается внутри тега *data*. Определение каждой переменной производится с помощью тега *variable*. У этого тега имеется атрибут *description*, содержащий описание соответствующей переменной. Можно также задать функцию получения словесного описания текущего состояния модели данных. Для этого служит тег *toString*.

Описание реализации алгоритма на языке *XML* изоморфно алгоритмической части программы на языке *Java*, приведенной в Приложении 2.

В описании реализации алгоритма оператор присваивания представляется знаком «@=», оператор ветвления – тегом *if*, цикл с предусловием – тегом *while*. У перечисленных тегов есть атрибуты, которые, в частности, содержат текстовые описания вызываемых операторов, используемые при составлении комментариев в автоматически генерируемых файлах реализации, а также комментарии для понимания выполняемого шага алгоритма.

Действия, которые необходимо выполнять при работе алгоритма, локализуются внутри тегов *action*. Они, в свою очередь, размещаются в тегах *step*, отмечающих шаги алгоритма, которые требуется визуализировать, заостряя на них внимание пользователя. Важнейшим атрибутом этого тега является атрибут *level*, хранящий целую величину, называемую «интересностью» данного шага. Когда «интересность» достигает порогового значения, то данный шаг будет отображен визуализатором. В противном случае данный шаг не будет выделяться визуализатором, и программа сразу перейдет к содержимому следующего тега *step*.

В *XML* - описании алгоритма также должен использоваться тег *draw*, содержащий вызов метода визуализатора, выполняющего перерисовку графического изображения массива.

Так как дальнейшая реализация алгоритма осуществляется на основе автоматного подхода, все теги, которые отвечают за функционирование одного и того же автомата, собираются в одном теге *auto*. Эти теги, вместе с тегом data, вкладываются в тег *algorithm*.

Третий файл *RadixSort-Configuration.xml* содержит параметры настройки интерфейса визуализатора,

6. Описание интерфейса визуализатора

На рис. 1 приведен внешний вид визуализатора.

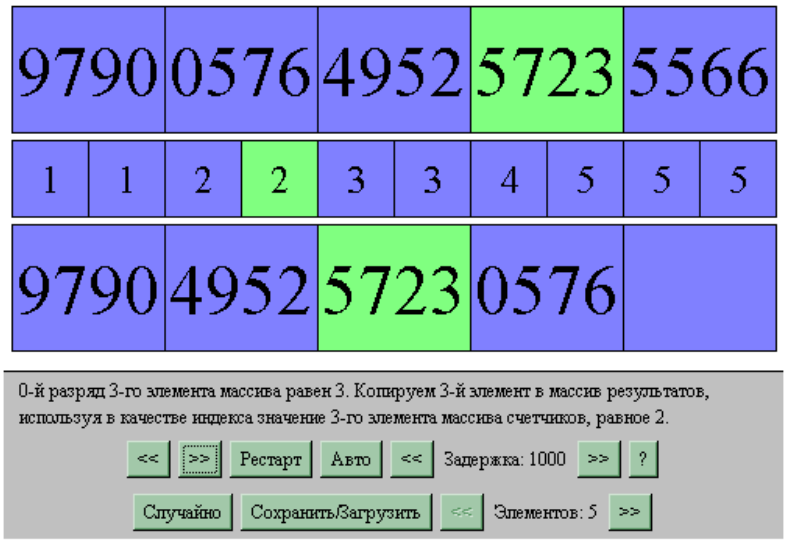

Рис. 1. Внешний вид визуализатора

Интерфейс визуализатора состоит из трех частей. Первый уровень – область визуализации, второй уровень – комментарии к соответствующему шагу алгоритма, а третий уровень – панель управления.

Область визуализации состоит из трех слоев:

- массив сортируемых чисел;
- массив счетчиков;
- вспомогательный массив.

Рассмотрим кнопки на этой панели слева направо и сверху вниз:

- кнопки «<<» и «>>» служат для перехода к следующему и предыдущему шагам алгоритма;
- кнопка «Рестарт» позволяет запустить процесс визуализации сначала;
- кнопка «Авто» включает автоматический режим. В нем переход к следующему шагу визуализации осуществляется через равные промежутки времени, длительность которых можно регулировать. При повторном нажатии этой кнопки (которая в

автоматическом режиме меняет название на «Стоп»), осуществляется выход из автоматического режима;

- второй набор кнопок «<<» и «>>» позволяет регулировать задержку между шагами алгоритма в автоматическом режиме, величина которой отображается в текстовой области, расположенной между ними. Рассмотренная совокупность “элементов” называется *спин-панелью*;
- кнопка «?» отображает окно с информацией о визуализаторе алгоритма, например об его авторе;
- кнопка «Случайно» позволяет заполнить сортируемый массив псевдослучайными числами;
- нижняя *спин-панель* обеспечивает возможность регулирования числа элементов в сортируемом массиве.

Нажатие последних трех кнопок приводит к инициализации алгоритма. После этого процесс визуализации начинается сначала.

7. Автоматически сгенерированный исходный код, реализующий логику визуализатора

Рассмотрим исходный код, сгенерированный программой *Vizi* (Приложение 4). Этот код содержит класс *RadixSort*, содержащий, в свою очередь, классы *Data* – модель данных и *Main* – реализация пары автоматов для движения по алгоритму в «прямом» и «обратном» направлениях.

Каждый автомат реализуется двумя операторами *switch.* В автомате для прямого прохода первый оператор обеспечивает переход в следующее состояние, а второй – действия в текущем состоянии. В автомате для обратного прохода первый оператор обеспечивает обращение действий в текущем состоянии, а второй – переход в предыдущее состояние. Каждый из автоматов в данном случае содержит по 19 состояний.

В случае наличия в *XML* – описании нескольких тегов *auto* будет сгенерировано несколько аналогичных пар автоматов, имена которых определяются значением атрибута *id*.

Основными методами класса *Main* являются методы *stepForward* и *stepBackward*. Первый из них обеспечивает переход в следующее состояние автомата с выполнением соответствующих действий, а второй — переход в обратном направлении. При этом каждый из этих методов может совершить несколько шагов за один вызов, в зависимости от того, считается ли данное состояние «интересным». «Интересность» состояния определяется методом *isInteresting*, использующим значение атрибута *level* тега *step*. Вывод комментариев к состояниям осуществляется методом *getComment*, а отрисовка состояний методом *drawState*.

Привязка к состояниям комментариев и действий по отрисовке также осуществляется автоматически с помощью дополнительных операторов *switch*.

8. Исходный код интерфейса визуализатора

В Приложений 5 приведен исходный код интерфейса визуализатора, написанный вручную на языке *Java* с использованием библиотеки *Vizi.* Для конфигурации внешнего вида интерфейса используются параметры, задаваемые в файле *RadixSort-Configuration.xml.* Все параметры конфигурации указываются внутри тега *configuration*.

С помощью тега *property* можно изменить ключевые параметры интерфейса визуализатора такие, как например, высота области для комментариев (параметр *comment-height*), максимальное значение элемента массива, присваиваемое генератором случайных чисел (параметр *max-value*).

Кнопки, расположенные во втором ряду панели управления визуализатора, не входят в набор стандартных кнопок для управления процессом визуализации произвольных алгоритмов и добавляются в файле *RadixSort-Configuration.xml*, описывающем интерфейс рассматриваемого визуализатора.

Вследствие того, что входными данными для визуализируемого алгоритма является массив, необходимо предоставить пользователю возможность изменять его длину, а также заполнять его псевдослучайными числами.

Для изменения размера массива, как отмечалось выше, создается *спин-панель*. Для ее создания используется тег *spin-panel*, атрибуты которого позволяют устанавливать названия кнопок, подсказки (*hints*), а также минимальное и максимальное значение числа, которое панель позволяет задать.

Для описания кнопки «Случайно» используется тег *button* с параметром *button-random*.

Технология *Vizi* предоставляет возможность изменять цветовую гамму визуализатора без перекомпиляции апплета. Для этого предусмотрен механизм таблиц стилей (*stylesheets*).

Заключение

Отметим ряд преимуществ использования технологии *Vizi* перед «классической» реализацией визуализаторов [3]:

- построение по *XML* – описанию алгоритма не только его прямого прохода, но и обратного, каждый из которых реализуется двумя операторами *switch*;
- описание алгоритма при помощи языка *XML* позволяет автоматически вводить комментарии в код, что значительно повышает его читабельность, а следовательно упрощает дальнейшее сопровождение;
- привязка к состояниям комментариев и действий по отрисовке также осуществляется автоматически с помощью дополнительных операторов *switch*;
- использование для построения визуализаторов единой технологии стандартизирует процесс разработки и позволяет во многих случаях избежать дублирования кода, уменьшая вероятность появления ошибок;
- стандартный эргономичный интерфейс визуализаторов имеет большое значение и обуславливает удобство изучения коллекций визуализаторов алгоритмов;
- легкость модификации *XML*-описания и автоматическое (!) изменение обратного прохода;
- компактность *XML*-описания – семь страниц этого описания против 16-и страниц соответствующего (автоматически сгенерированного) исходного кода.

Литература

1. *Казаков М.А., Корнеев Г.А., Шалыто А.А.* Разработка логики визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6.
2. *Кнут Д.Е.* Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. М.: Мир, 1978.
3. http://rain.ifmo.ru/cat
4. *Шалыто А.А.* Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9. http://is.ifmo.ru раздел «Статьи».

Приложение 1. Исходный текст реализации алгоритма поразрядной сортировки

```
class RadixSort_Orig {

    public static void main (String args []) {
        int a[] = new int[]{7834, 345, 55, 2343, 5466, 4355,2221, 544, 3,
5667,99,3333,5465, 5656,1233,78,554};

        int digit;
        final int N = 4;
        final int RADIX = 10;

        int p = 1;
        for(int i = 0; i < N; i++){
            int counter[] = new int[RADIX];
            int res[] = new int[a.length];

            for(int j = 0; j < a.length; j++){
                counter[a[j] / p % RADIX]++;
            }

            int v = 0;
            for(int j = 0; j < RADIX; j++){
                v += counter[j];
                counter[j] = v - counter[j];
            }

            for(int j = 0; j < a.length; j++){
                digit = a[j] / p % RADIX;
                res[counter[digit]] = a[j];
                counter[digit]++;
            }

            a = res;
            p *= RADIX;
        }
        for(int i = 0; i < a.length; i++){
            System.out.println(a[i]);
        }
    }
}
```

Приложение 2. Исходный текст преобразованной реализации алгоритма

```
class RadixSort {

    public static void main (String args []) {
        int a[] = new int[]{7834, 345, 55, 2343, 5466, 4355,2221, 544, 3,
5667,99,3333,5465, 5656,1233,78,554};

        int digit;
        final int N = 4;
        final int RADIX = 10;

        int p = 1;
        int i = 0;
        while (i < N) {
            int counter[] = new int[RADIX];
            int res[] = new int[a.length];

            int j = 0;
            while (j < a.length) {
                counter[a[j] / p % RADIX]++;
                j++;
            }

            int v = 0;
            j = 0;
            while (j < RADIX) {
                v += counter[j];
                counter[j] = v - counter[j];
                j++;
            }

            j = 0;
            while (j < a.length) {
                digit = a[j] / p % RADIX;
                res[counter[digit]] = a[j];
                counter[digit]++;
                j++;
            }

            a = res;
            p *= RADIX;
            i++;
        }
    }
}
```

Приложение 3. *XML*-описания визуализатора

Все *XML*-описания алгоритма пишутся разработчиком вручную по тексту преобразованной программы. По ним автоматически генерируются файла реализации алгоритмов.

Файл *RadixSort.xml*

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

<!--
    "RadixSort" visualizer description
    Version: $Id: RadixSort.xml,v 1.0 2003/11/18 $
-->

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

<visualizer
    id="RadixSort"
    package="ru.ifmo.vizi.RadixSort"
    main-class="RadixSortVisualizer"

    preferred-width="500"
    preferred-height="350"

    name-ru= "Поразрядная сортировка набора чисел"
    name-en="RadixSort"

    author-ru="Антон Пономарев"
    author-en="Anton Ponomarev"
    author-email="ponom@rain.ifmo.ru"

    supervisor-ru="Анатолий Шалыто"
    supervisor-en="Anatoly Shalyto"
```

```
    supervisor-email="shalyto@mail.ifmo.ru"

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

Файл *RadixSort-Algorithm.xml*

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

<!--
    "RadixSort" algoritm description (using auto-reverse feature)
    Version: $Id: RadixSort-Algorithm.xml,v 1.0 2003/11/18 $
-->

<algorithm>
        <variable
            description="Массив для сортировки"
            type="int[]"
            name="a"
            value="new int[]{7834, 345, 55, 2343, 5466}"/>
        <variable
            description="Результирующий массив"
            type="int[]"
            name="res"
            value="new int[]{0, 0, 0, 0, 0}"/>
        <variable
            description="Массив счетчиков"
            type="int[]"
            name="counter"
            value="new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}"/>
        <variable
            description="Экземпляр апплета"
            type="RadixSortVisualizer"
            name="visualizer"
```

```
            value="null"/>
    <data>
        <toString>
            StringBuffer s = new StringBuffer();
            return s.toString();
        </toString>
    </data>

    <auto id="Main" description="Производит поразрядную сортировку массива натуральных чисел">

        <variable
            description="Переменная цикла по разрядам"
            type="int"
            name="i"
            />
        <variable
            description="Переменная цикла по элементам массива"
            type="int"
            name="j"
            />

        <variable
            description="Переменная для хранения текущей цифры"
            type="int"
            name="digit"
            />

        <variable
            description="Переменная для хранения суммы значения счетчиков"
            type="int"
            name="v"
        />

        <variable
            description="Текущая степень 10"
            type="int"
            name="p"
        />
```

```
<start
    comment-ru="На экране изображен массив который будет отсортирован"
    comment-en="There is an array on the display"
>
    <draw>
        @visualizer.updateArray(-1, -1, -1);
    </draw>
</start>
<step
    id="Initialization"
    description="Инициализация"
    comment-ru="Инициализируем."
    comment-en="Intitialize."
>
    <draw>
        @visualizer.updateArray(-1, -1, -1);
    </draw>
    <action>
        @i @= 0;
    </action>
</step>
<step
    id="LoopInit1"
    description="Начало цикла по разрядам"
    level="-1"
>
    <action>
        @i @= 0;
        @p @= 1;
    </action>
</step>
<while
    id="Loop1"
    description="Цикл по разрядам"
    test="@i &lt; 4"
    level="-1"
>
    <step
        id="LoopInit2"
        description="Начало цикла по элементам"
```

```
            level="-1"
        >
            <action>
                @counter @= new int[10];
                @res @= new int[@a.length];
                @j @= 0;
            </action>
        </step>
        <while
            id="Loop2"
            description="Цикл по элементам"
            test="@j &lt; @a.length"
            level="-1"
        >
            <step
                id="inccounter"
                description="Инкремент counter"
                level="0"
                comment-ru="{0}-й разряд {1}-го элемента массива равен {2}.
Увеличиваем {2}-й элемент массива счетчиков на 1."
                comment-en="Digit {0} of array element number {1} equals
{2}. Increase counter array element number {2} by 1."
                comment-args="new Integer(@i), new Integer(@j), new
Integer(@digit)"
            >
                <draw>
                    @visualizer.updateArray(@j, @digit, -1);
                </draw>
                <action>
                    @digit @= @a[@j] / @p % 10;
                    @counter[@a[@j] / @p % 10] @= @counter[@a[@j] / @p % 10]
                          + 1;
                </action>
            </step>

            <step
                id="incj2"
                description="Инкремент j"
                level="-1"
            >
```

```
                <action>
                    @j @= @j + 1;
                </action>
            </step>

        </while>


        <step
            id="LoopInit3"
            description="Начало цикла по элементам массива счетчиков"
            level="-1"
        >
            <action>
                @j @= 0;
                @v @= 0;
            </action>
        </step>

        <while
            id="Loop3"
            description="Цикл по элементам массива счетчиков"
            test="@j &lt; 10"
            level="-1"
        >
            <step
                id="sumcounter"
                description="Суммирование счетчика"
                level="0"
                comment-ru="В {0}-й элемент массива счетчиков записываем
сумму всех предыдущих элементов."
                comment-en="Put sum of all previous elements into counter
array element number {0}."
                comment-args="new Integer(@j)"
            >
                <draw>
                    @visualizer.updateArray(-1, @j + 1, -1);
                </draw>
                <action>
                    @v @= @v + @counter[@j];
```

```
                @counter[@j] @= @v - @counter[@j];
            </action>
        </step>

        <step
            id="incj3"
            description="Инкремент j"
            level="-1"
        >
            <action>
                @j @= @j + 1;
            </action>
        </step>

    </while>


    <step
        id="LoopInit4"
        description="Начало цикла по элементам"
        level="-1"
    >
        <action>
            @j @= 0;
        </action>
    </step>

    <while
        id="Loop4"
        description="Цикл по элементам массива"
        test="@j &lt; @a.length"
        level="-1"
    >
        <step
            id="fillresult"
            description="Копирование элемента"
            level="0"
            comment-ru="{0}-й разряд {1}-го элемента массива равен {2}.
Копируем {1}-й элемент в массив результатов, используя в качестве индекса
значение {2}-го элемента массива счетчиков, равное {3}."
```

```
                    comment-en="Digit {0} of array element number {1} equals
{2}. Copy element number {1} into results array using value of counter array
element number {2}, that equals {3} as index."
                    comment-args="new Integer(@i), new Integer(@j), new
Integer(@digit), new Integer(@counter[@digit])"
                >
                    <draw>
                        @visualizer.updateArray(@j, @digit, @counter[@digit]);
                    </draw>

                    <action>
                        @digit @= @a[@j] / @p % 10;
                        @res[@counter[@digit]] @= @a[@j];
                    </action>
                </step>
                <step
                    id="inccounter2"
                    description="Увеличение счетчика"
                    level="0"
                    comment-ru="Увеличиваем значение {0}-го элемента массива
                        счетчиков на 1."
                    comment-en="Increase value of counter array element number
                        {0} by 1."
                    comment-args="new Integer(@digit)"
                >
                    <draw>
                        @visualizer.updateArray(-1, @digit, -1);
                    </draw>

                    <action>
                        @counter[@digit] @= @counter[@digit] + 1;
                    </action>
                </step>

                <step
                    id="incj4"
                    description="Инкремент j"
                    level="-1"
                >
                    <action>
```

```
                    @j @= @j + 1;
                </action>
            </step>

        </while>

        <step
            id="arrayexch"
            description="Перемещение результата в исходный массив"
            level="-1"
        >
            <action>
                @a @= @res;
            </action>
        </step>

        <step
            id="inci"
            description="Инкремент i"
            level="-1"
        >
            <action>
                @p @= @p * 10;
                @i @= @i + 1;
            </action>
        </step>
    </while>
        <finish
            comment-ru="Сортировка завершена."
            comment-en="Sorting finished."
        >
            <draw>
                @visualizer.updateArray(-1, -1, -1);
            </draw>
        </finish>
    </auto>
</algorithm>
```

Файл *RadixSort-Configuration.xml*

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

<!--
   "RadixSort" visualizer configuration (example).
   Version: $Id: RadixSort-Configuration.xml,v 1.2 2003/12/25 12:57:27 geo Exp $
-->

<configuration>
    <property
        description = "Comment pane height"
        param       = "comment-height"
        value       = "40"
    />
    <spin-panel
        description = "Number of elements in the array"
        param       = "elements"
        caption-ru  = "Элементов: {0,number,####}"
        caption-en  = "Elements: {0,number,####}"
        hint-ru     = "Количество элементов в массиве"
        hint-en     = "Number of elements in the array"
        value       = "5"
        min-value   = "5"
        max-value   = "20"
        step        = "1"
    >
        <button
            param       = "button-less"
            caption-ru  = "&lt;&lt;"
            caption-en  = "&lt;&lt;"
            hint-ru     = "Уменьшить количество элементов"
            hint-en     = "Decrease number of elements"
        />
        <button
            param       = "button-more"
            caption-ru  = ">>"
            caption-en  = ">>"
            hint-ru     = "Увеличить количество элементов"
```

```
        hint-en     = "Increase number of elements"
    />
</spin-panel>
<button
    description = "Fills the array with random values"
    param       = "button-random"
    caption-ru  = "Случайно"
    caption-en  = "Random"
    hint-ru     = "Заполнить массив случайными значениями"
    hint-en     = "Fill the array with random values"
/>
<message
    description = 'Comment for "ArrayLength" parameter in the output file'
    param       = "ArrayLengthComment"
    message-ru  = "Длина массива ({0} ... {1})"
    message-en  = "Array length ({0} ... {1})"
/>
<message
    description = 'Comment for "Elements" parameter in the output file'
    param       = "ElementsComment"
    message-ru  = "Элементы массива ({0} ... {1})"
    message-en  = "Array elements ({0} ... {1})"
/>
<message
    description = 'Comment for "Step" parameter in the output file'
    param       = "StepComment"
    message-ru  = "Номер шага"
    message-en  = "Current step"
/>
<styleset
    description = "Array style set"
    param       = "array"
>
    <style
        description     = "Ordinary cell"
        text-color      = "000000"
        text-align      = "0.5"
        border-color    = "000000"
        border-status   = "true"
        fill-color      = "8080ff"
```

```
        fill-status     = "true"
        aspect-status   = "false"
        padding         = "0.2"
    >
        <font
            face        = "Serif"
            size        = "12"
            style       = "plain"
        />
    </style>
    <style
        description     = "Selected cell"
        fill-color      = "80ff80"
    />
    <style
        description     = "Local-maximum cell"
        fill-color      = "ff8080"
    />
</styleset>
<property
    description = "Maximum possible value of the element"
    param       = "max-value"
    value       = "9999"
/>
<property
    description = "Widest possible value of the element"
    param       = "max-value-string"
    value       = "9999"
/>
<group
    description = "Save/Load dialog configuration"
    param       = "SaveLoadDialog"
>
    <property
        description = "Height of the comment pane"
        param       = "CommentPane-lines"
        value       = "2"
    />
    <property
        description = "Width of the text area"
```

```
            param       = "columns"
            value       = "40"
        />
        <property
            description = "Height of the text area"
            param       = "rows"
            value       = "7"
        />
    </group>
</configuration>
```

Приложение 4. Автоматически сгенерированный исходный код

```
package ru.ifmo.vizi.RadixSort;

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

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

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

    /**
      * Данные.
      */
    public final class Data {
        /**
          * Массив для сортировки.
          */
        public int[] a = new int[]{7834, 345, 55, 2343, 5466};

        /**
          * Результирующий массив.
          */
        public int[] res = new int[]{0, 0, 0, 0, 0};

        /**
          * Массив счетчиков.
          */
        public int[] counter = new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

        /**
          * Экземпляр апплета.
          */
        public RadixSortVisualizer visualizer = null;

        /**
          * Переменная цикла по разрядам (Процедура Main).
          */
        public int Main_i;

        /**
          * Переменная цикла по элементам массива (Процедура Main).
          */
        public int Main_j;

        /**
```

```
      * Переменная для хранения текущей цифры (Процедура Main).
      */
    public int Main_digit;

    /**
      * Переменная для хранения суммы значения счетчиков (Процедура Main).
      */
    public int Main_v;

    /**
      * Текущая степень 10 (Процедура Main).
      */
    public int Main_p;

    public String toString() {
        StringBuffer s = new StringBuffer();
        return s.toString();
    }
}

/**
  * Производит поразрядную сортировку массива натуральных чисел.
  */
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 = 19;

    /**
      * Конструктор.
      */
    public Main() {
        super(
            "Main",
            0, // Номер начального состояния
            19, // Номер конечного состояния
            new String[]{
                "Начальное состояние",
                "Инициализация",
                "Начало цикла по разрядам",
                "Цикл по разрядам",
                "Начало цикла по элементам",
                "Цикл по элементам",
                "Инкремент counter",
                "Инкремент j",
                "Начало цикла по элементам массива счетчиков",
                "Цикл по элементам массива счетчиков",
                "Суммирование счетчика",
                "Инкремент j",
                "Начало цикла по элементам",
                "Цикл по элементам массива",
                "Копирование элемента",
                "Увеличение счетчика",
```

```
            "Инкремент j",
            "Перемещение результата в исходный массив",
            "Инкремент i",
            "Конечное состояние"
        }, new int[]{
            Integer.MAX_VALUE, // Начальное состояние,
            0, // Инициализация
            -1, // Начало цикла по разрядам
            -1, // Цикл по разрядам
            -1, // Начало цикла по элементам
            -1, // Цикл по элементам
            0, // Инкремент counter
            -1, // Инкремент j
            -1, // Начало цикла по элементам массива счетчиков
            -1, // Цикл по элементам массива счетчиков
            0, // Суммирование счетчика
            -1, // Инкремент j
            -1, // Начало цикла по элементам
            -1, // Цикл по элементам массива
            0, // Копирование элемента
            0, // Увеличение счетчика
            -1, // Инкремент j
            -1, // Перемещение результата в исходный массив
            -1, // Инкремент i
            Integer.MAX_VALUE, // Конечное состояние
        }
    );
}

/**
  * Сделать один шаг автомата вперед.
  */
protected void doStepForward(int level) {
    // Переход в следующее состояние
    switch (state) {
        case START_STATE: { // Начальное состояние
            state = 1; // Инициализация
            break;
        }
        case 1: { // Инициализация
            state = 2; // Начало цикла по разрядам
            break;
        }
        case 2: { // Начало цикла по разрядам
            stack.pushBoolean(false);
            state = 3; // Цикл по разрядам
            break;
        }
        case 3: { // Цикл по разрядам
            if (d.Main_i < 4) {
                state = 4; // Начало цикла по элементам
            } else {
                state = END_STATE;
            }
            break;
        }
        case 4: { // Начало цикла по элементам
            stack.pushBoolean(false);
            state = 5; // Цикл по элементам
```

```
    break;
}
case 5: { // Цикл по элементам
    if (d.Main_j < d.a.length) {
        state = 6; // Инкремент counter
    } else {
        state = 8; // Начало цикла по элементам массива
                   // счетчиков
    }
    break;
}
case 6: { // Инкремент counter
    state = 7; // Инкремент j
    break;
}
case 7: { // Инкремент j
    stack.pushBoolean(true);
    state = 5; // Цикл по элементам
    break;
}
case 8: { // Начало цикла по элементам массива счетчиков
    stack.pushBoolean(false);
    state = 9; // Цикл по элементам массива счетчиков
    break;
}
case 9: { // Цикл по элементам массива счетчиков
    if (d.Main_j < 10) {
        state = 10; // Суммирование счетчика
    } else {
        state = 12; // Начало цикла по элементам
    }
    break;
}
case 10: { // Суммирование счетчика
    state = 11; // Инкремент j
    break;
}
case 11: { // Инкремент j
    stack.pushBoolean(true);
    state = 9; // Цикл по элементам массива счетчиков
    break;
}
case 12: { // Начало цикла по элементам
    stack.pushBoolean(false);
    state = 13; // Цикл по элементам массива
    break;
}
case 13: { // Цикл по элементам массива
    if (d.Main_j < d.a.length) {
        state = 14; // Копирование элемента
    } else {
        state = 17; // Перемещение результата в исходный массив
    }
    break;
}
case 14: { // Копирование элемента
    state = 15; // Увеличение счетчика
    break;
}
```

```
    case 15: { // Увеличение счетчика
        state = 16; // Инкремент j
        break;
    }
    case 16: { // Инкремент j
        stack.pushBoolean(true);
        state = 13; // Цикл по элементам массива
        break;
    }
    case 17: { // Перемещение результата в исходный массив
        state = 18; // Инкремент i
        break;
    }
    case 18: { // Инкремент i
        stack.pushBoolean(true);
        state = 3; // Цикл по разрядам
        break;
    }
}

// Действие в текущем состоянии
switch (state) {
    case 1: { // Инициализация
        startSection();
        storeField(d, "Main_i");
        d.Main_i = 0;
        break;
    }
    case 2: { // Начало цикла по разрядам
        startSection();
        storeField(d, "Main_i");
        d.Main_i = 0;
        storeField(d, "Main_p");
        d.Main_p = 1;
        break;
    }
    case 3: { // Цикл по разрядам
        break;
    }
    case 4: { // Начало цикла по элементам
        startSection();
        storeField(d, "counter");
        d.counter = new int[10];
        storeField(d, "res");
        d.res = new int[d.a.length];
        storeField(d, "Main_j");
        d.Main_j = 0;
        break;
    }
    case 5: { // Цикл по элементам
        break;
    }
    case 6: { // Инкремент counter
        startSection();
        storeField(d, "Main_digit");
        d.Main_digit = d.a[d.Main_j] / d.Main_p % 10;
        storeArray(d.counter, d.a[d.Main_j] / d.Main_p % 10);
        d.counter[d.a[d.Main_j] / d.Main_p % 10] =
            d.counter[d.a[d.Main_j] / d.Main_p % 10] + 1;
```

```
    break;
}
case 7: { // Инкремент j
    startSection();
    storeField(d, "Main_j");
    d.Main_j = d.Main_j + 1;
    break;
}
case 8: { // Начало цикла по элементам массива счетчиков
    startSection();
    storeField(d, "Main_j");
    d.Main_j = 0;
    storeField(d, "Main_v");
    d.Main_v = 0;
    break;
}
case 9: { // Цикл по элементам массива счетчиков
    break;
}
case 10: { // Суммирование счетчика
    startSection();
    storeField(d, "Main_v");
    d.Main_v = d.Main_v + d.counter[d.Main_j];
    storeArray(d.counter, d.Main_j);
    d.counter[d.Main_j] = d.Main_v - d.counter[d.Main_j];
    break;
}
case 11: { // Инкремент j
    startSection();
    storeField(d, "Main_j");
    d.Main_j = d.Main_j + 1;
    break;
}
case 12: { // Начало цикла по элементам
    startSection();
    storeField(d, "Main_j");
    d.Main_j = 0;
    break;
}
case 13: { // Цикл по элементам массива
    break;
}
case 14: { // Копирование элемента
    startSection();
    storeField(d, "Main_digit");
    d.Main_digit = d.a[d.Main_j] / d.Main_p % 10;
    storeArray(d.res, d.counter[d.Main_digit]);
    d.res[d.counter[d.Main_digit]] = d.a[d.Main_j];
    break;
}
case 15: { // Увеличение счетчика
    startSection();
    storeArray(d.counter, d.Main_digit);
    d.counter[d.Main_digit] = d.counter[d.Main_digit] + 1;
    break;
}
case 16: { // Инкремент j
    startSection();
    storeField(d, "Main_j");
```

```
            d.Main_j = d.Main_j + 1;
            break;
        }
        case 17: { // Перемещение результата в исходный массив
            startSection();
            storeField(d, "a");
            d.a = d.res;
            break;
        }
        case 18: { // Инкремент i
            startSection();
            storeField(d, "Main_p");
            d.Main_p = d.Main_p * 10;
            storeField(d, "Main_i");
            d.Main_i = d.Main_i + 1;
            break;
        }
    }
}

/**
  * Сделать один шаг автомата назад.
  */
protected void doStepBackward(int level) {
    // Обращение действия в текущем состоянии
    switch (state) {
        case 1: { // Инициализация
            restoreSection();
            break;
        }
        case 2: { // Начало цикла по разрядам
            restoreSection();
            break;
        }
        case 3: { // Цикл по разрядам
            break;
        }
        case 4: { // Начало цикла по элементам
            restoreSection();
            break;
        }
        case 5: { // Цикл по элементам
            break;
        }
        case 6: { // Инкремент counter
            restoreSection();
            break;
        }
        case 7: { // Инкремент j
            restoreSection();
            break;
        }
        case 8: { // Начало цикла по элементам массива счетчиков
            restoreSection();
            break;
        }
        case 9: { // Цикл по элементам массива счетчиков
            break;
        }
```

```
    case 10: { // Суммирование счетчика
        restoreSection();
        break;
    }
    case 11: { // Инкремент j
        restoreSection();
        break;
    }
    case 12: { // Начало цикла по элементам
        restoreSection();
        break;
    }
    case 13: { // Цикл по элементам массива
        break;
    }
    case 14: { // Копирование элемента
        restoreSection();
        break;
    }
    case 15: { // Увеличение счетчика
        restoreSection();
        break;
    }
    case 16: { // Инкремент j
        restoreSection();
        break;
    }
    case 17: { // Перемещение результата в исходный массив
        restoreSection();
        break;
    }
    case 18: { // Инкремент i
        restoreSection();
        break;
    }
}

// Переход в предыдущее состояние
switch (state) {
    case 1: { // Инициализация
        state = START_STATE;
        break;
    }
    case 2: { // Начало цикла по разрядам
        state = 1; // Инициализация
        break;
    }
    case 3: { // Цикл по разрядам
        if (stack.popBoolean()) {
            state = 18; // Инкремент i
        } else {
            state = 2; // Начало цикла по разрядам
        }
        break;
    }
    case 4: { // Начало цикла по элементам
        state = 3; // Цикл по разрядам
        break;
    }
```

```
case 5: { // Цикл по элементам
    if (stack.popBoolean()) {
        state = 7; // Инкремент j
    } else {
        state = 4; // Начало цикла по элементам
    }
    break;
}
case 6: { // Инкремент counter
    state = 5; // Цикл по элементам
    break;
}
case 7: { // Инкремент j
    state = 6; // Инкремент counter
    break;
}
case 8: { // Начало цикла по элементам массива счетчиков
    state = 5; // Цикл по элементам
    break;
}
case 9: { // Цикл по элементам массива счетчиков
    if (stack.popBoolean()) {
        state = 11; // Инкремент j
    } else {
        state = 8; // Начало цикла по элементам массива
                   // счетчиков
    }
    break;
}
case 10: { // Суммирование счетчика
    state = 9; // Цикл по элементам массива счетчиков
    break;
}
case 11: { // Инкремент j
    state = 10; // Суммирование счетчика
    break;
}
case 12: { // Начало цикла по элементам
    state = 9; // Цикл по элементам массива счетчиков
    break;
}
case 13: { // Цикл по элементам массива
    if (stack.popBoolean()) {
        state = 16; // Инкремент j
    } else {
        state = 12; // Начало цикла по элементам
    }
    break;
}
case 14: { // Копирование элемента
    state = 13; // Цикл по элементам массива
    break;
}
case 15: { // Увеличение счетчика
    state = 14; // Копирование элемента
    break;
}
case 16: { // Инкремент j
    state = 15; // Увеличение счетчика
```

```
            break;
        }
        case 17: { // Перемещение результата в исходный массив
            state = 13; // Цикл по элементам массива
            break;
        }
        case 18: { // Инкремент i
            state = 17; // Перемещение результата в исходный массив
            break;
        }
        case END_STATE: { // Начальное состояние
            state = 3; // Цикл по разрядам
            break;
        }
    }
}

/**
  * Комментарий к текущему состоянию
  */
public String getComment() {
    String comment = "";
    Object[] args = null;
    // Выбор комментария
    switch (state) {
        case START_STATE: { // Начальное состояние
            comment = RadixSort.this.getComment("Main.START_STATE");
            break;
        }
        case 1: { // Инициализация
            comment = RadixSort.this.getComment("Main.Initialization");
            break;
        }
        case 6: { // Инкремент counter
            comment = RadixSort.this.getComment("Main.inccounter");
            args = new Object[]{new Integer(d.Main_i), new
                Integer(d.Main_j), new Integer(d.Main_digit)};
            break;
        }
        case 10: { // Суммирование счетчика
            comment = RadixSort.this.getComment("Main.sumcounter");
            args = new Object[]{new Integer(d.Main_j)};
            break;
        }
        case 14: { // Копирование элемента
            comment = RadixSort.this.getComment("Main.fillresult");
            args = new Object[]{new Integer(d.Main_i), new
                Integer(d.Main_j), new Integer(d.Main_digit), new
                Integer(d.counter[d.Main_digit])};
            break;
        }
        case 15: { // Увеличение счетчика
            comment = RadixSort.this.getComment("Main.inccounter2");
            args = new Object[]{new Integer(d.Main_digit)};
            break;
        }
        case END_STATE: { // Конечное состояние
            comment = RadixSort.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.visualizer.updateArray(-1, -1, -1);
                    break;
                }
                case 1: { // Инициализация
                    d.visualizer.updateArray(-1, -1, -1);
                    break;
                }
                case 6: { // Инкремент counter
                    d.visualizer.updateArray(d.Main_j, d.Main_digit, -1);
                    break;
                }
                case 10: { // Суммирование счетчика
                    d.visualizer.updateArray(-1, d.Main_j, -1);
                    break;
                }
                case 14: { // Копирование элемента
                    d.visualizer.updateArray(d.Main_j, d.Main_digit,
                        d.counter[d.Main_digit]);
                    break;
                }
                case 15: { // Увеличение счетчика
                    d.visualizer.updateArray(-1, d.Main_digit, -1);
                    break;
                }
                case END_STATE: { // Конечное состояние
                    d.visualizer.updateArray(-1, -1, -1);
                    break;
                }
            }
        }
    }
}
```

Приложение 5. Исходный код интерфейса визуализатора

```
package ru.ifmo.vizi.RadixSort;

import ru.ifmo.vizi.base.ui.*;
import ru.ifmo.vizi.base.*;
import ru.ifmo.vizi.base.widgets.Rect;
import ru.ifmo.vizi.base.widgets.ShapeStyle;

import java.awt.*;
import java.util.Stack;

/**
 * Radix Sort applet.
 *
 * @author  Georgiy Korneev
 * @version $Id: RadixSortVisualizer.java,v 1.4 2004/01/23 14:40:04 geo Exp $
 */
public final class RadixSortVisualizer extends Base {
    /**
     * Radix Sort automata instance.
     */
    private final RadixSort auto;

    /**
     * Radix Sort automata data.
     */
    private final RadixSort.Data data;

    /**
     * Cells with array elements.
     * Vector of {@link Rect}.
     */
    private final Stack cells;
    private final Stack cells_b;
    private final Stack cells_c;

    /**
     * Number of elements in array.
```

```
 */
private final SpinPanel elements;

/**
 * Maximal array value.
 */
private final int maxValue;

/**
 * Maximal array value string.
 */
private final String maxValueString;


/**
 * Array shape style set.
 */
private final ShapeStyle[] styleSet;

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

/**
 * Creates a new RadixSort  visualizer.
 *
 * @param parameters visualizer parameters.
 */
public RadixSortVisualizer(VisualizerParameters parameters) {
    super(parameters);
    auto = new RadixSort(locale);
    data = auto.d;
    data.visualizer = this;
    cells = new Stack();
    cells_b = new Stack();
    cells_c = new Stack();

    styleSet = ShapeStyle.loadStyleSet(config, "array");
```

```
    elements = new SpinPanel(config, "elements") {
        protected void click(double value) {
            setArraySize(getIntValue());
        }
    };

    maxValue = config.getInteger("max-value");
    maxValueString = config.getParameter("max-value-string",
        Integer.toString(maxValue));
    setArraySize(elements.getIntValue());

    createInterface(auto);
}

/**
 * This method creates panel with visualizer controls.
 *
 * @return controls pane.
 */
public Component createControlsPane() {
    Panel panel = new Panel(new BorderLayout());

    panel.add(new AutoControlsPane(config, auto, forefather, false),
        BorderLayout.CENTER);

    Panel bottomPanel = new Panel();
    bottomPanel.add(new HintedButton(config, "button-random"){
        protected void click() {
            randomize();
        }
    });
    if (config.getBoolean("button-ShowSaveLoad")) {
        bottomPanel.add(new HintedButton(config, "button-SaveLoad") {
            protected void click() {
                saveLoadDialog.center();
                StringBuffer buffer = new StringBuffer();
                int[] a = auto.d.a;
                buffer.append("/* ").append(
                    I18n.message(
                        config.getParameter("ArrayLengthComment"),
```

```
                    new Double(elements.getMinValue()),
                    new Double(elements.getMaxValue())
                )
            ).append(" */\n");

            buffer.append("ArrayLength =
                ").append(a.length).append("\n");

            buffer.append("/* ").append(
                I18n.message(
                    config.getParameter("ElementsComment"),
                    new Integer(0),
                    new Integer(maxValue)
                )
            ).append(" */\n");

            buffer.append("Elements = ");
            for (int i = 0; i < a.length; i++) {
                buffer.append(a[i]).append(" ");
            }

            buffer.append("\n/* ").append(
                config.getParameter("StepComment")
            ).append(" */\n");
            buffer.append("Step = ").append(auto.getStep());
            saveLoadDialog.show(buffer.toString());
        }
    });
}
bottomPanel.add(elements);
panel.add(bottomPanel, BorderLayout.SOUTH);

saveLoadDialog = new SaveLoadDialog(config, forefather) {
    public boolean load(String text) throws Exception {
        SmartTokenizer tokenizer = new SmartTokenizer(text, config);
        tokenizer.expect("ArrayLength");
        tokenizer.expect("=");
        int[] a = new int[tokenizer.nextInt(
            (int) elements.getMinValue(),
            (int) elements.getMaxValue()
```

```
            )];

            tokenizer.expect("Elements");
            tokenizer.expect("=");
            for (int i = 0; i < a.length; i++) {
                a[i] = tokenizer.nextInt(0, maxValue);
            }

            tokenizer.expect("Step");
            tokenizer.expect("=");
            int step = tokenizer.nextInt();
            tokenizer.expectEOF();

            auto.d.a = a;
            rewind(step);

            return true;
        }
    };

    return panel;
}

/**
 * Adjusts array size to match current model size.
 */
private void adjustArraySize() {
    int size = auto.d.a.length;
    while (cells.size() < size) {
        Rect rect = new Rect(styleSet);
        cells.push(rect);
        clientPane.add(rect);
    }
    while (cells.size() > size) {
        clientPane.remove((Component) cells.pop());
    }

    while (cells_b.size() < size) {
        Rect rect = new Rect(styleSet);
        cells_b.push(rect);
```

```
            clientPane.add(rect);
        }
        while (cells_b.size() > size) {
            clientPane.remove((Component) cells_b.pop());
        }

        while (cells_c.size() < 10) {
            Rect rect = new Rect(styleSet);
            cells_c.push(rect);
            clientPane.add(rect);
        }
        while (cells_c.size() > 10) {
            clientPane.remove((Component) cells_c.pop());
        }


        clientPane.doLayout();
        elements.setValue(data.a.length);
    }

    /**
     * Sets new array size.
     *
     * @param size new array size.
     */
    private void setArraySize(int size) {
        auto.d.a = new int[size];
        randomize();
        adjustArraySize();
    }

    /**
     * Randomizes array values.
     */
    private void randomize() {
        for (int i = 0; i < data.a.length; i++) {
            data.a[i] = (int) (Math.random() * maxValue) + 1;
        }
        rewind(0);
    }
```

```
/**
 * Rewinds algorithm to the specified step.
 *
 * @param step spet of the algorith to rewind to.
 */
private void rewind(int step) {
    adjustArraySize();
    auto.toStart();

    updateArray(-1, -1, -1);
    while (!auto.isAtEnd() && auto.getStep() < step) {
        auto.stepForward(0);
    }
}

/**
 * Updates array view.
 *
 * @param activeCell current active cell.
 * @param activeStyle style of active cell.
 */
public void updateArray(int a_index, int counter_index, int res_index) {


    for (int i = 0; i < data.a.length; i++) {
        Rect rect = (Rect) cells.elementAt(i);
        StringBuffer s = new StringBuffer();
        int alength = Integer.toString(data.a[i]).length();
        for(int l = 0; l < 4 - alength; l++){
            s.append("0");
        }
        s.append(Integer.toString(data.a[i]));
        rect.setMessage(s.toString());
        rect.setStyle(i == a_index ? 1 : 0);
    }

    for (int i = 0; i < data.res.length; i++) {
        Rect rect = (Rect) cells_b.elementAt(i);
        if(data.res[i] > 0) {
```

```
            StringBuffer s = new StringBuffer();
            int blength = Integer.toString(data.res[i]).length();
            for(int l = 0; l < 4 - blength; l++){
            s.append("0");
            }
            s.append(Integer.toString(data.res[i]));
            rect.setMessage(s.toString());
            rect.setStyle(i == res_index ? 1 : 0);
        }else {
            rect.setMessage("");
            rect.setStyle(0);
        }

    }

    for (int i = 0; i < data.counter.length; i++) {
        Rect rect = (Rect) cells_c.elementAt(i);
        StringBuffer s = new StringBuffer();
        s.append(Integer.toString(data.counter[i]));
        rect.setMessage(s.toString());
        rect.setStyle(i == counter_index ? 1 : 0);

    }

    update(true);
}

/**
 * Invoked when client pane shoud be layouted.
 *
 * @param clientWidth client pane width.
 * @param clientHeight client pane height.
 */
protected void layoutClientPane(int clientWidth, int clientHeight) {
    int n = cells.size();


    double width = (clientWidth - 10.0) / n;
    double height = Math.min(width, (clientHeight) * 10.0 / 30.0);
    double y = height / 10;
```

```
        double x = (clientWidth - width * n) / 2;

        double counter_width = (clientWidth - 10) /10;
        double counter_height = Math.min(counter_width, (clientHeight) * 10.0 /
            30.0);

        for (int i = 0; i < n; i++) {
            Rect rect = (Rect) cells.elementAt(i);
            rect.setBounds((int)Math.round(x + i * width), (int)Math.round(y),
                  (int)Math.round(width + 1), (int)Math.round(height + 1));
            rect.adjustFontSize(maxValueString);
        }

        for (int i = 0; i < n; i++) {
            Rect rect = (Rect) cells_b.elementAt(i);
            rect.setBounds((int)Math.round(x + i * width), (int)Math.round(y +
                  height + counter_height + 10), (int)Math.round(width + 1),
                  (int)Math.round(height + 1));
            rect.adjustFontSize(maxValueString);
        }

        for (int i = 0; i < 10; i++) {
            Rect rect = (Rect) cells_c.elementAt(i);
            rect.setBounds((int)Math.round(x + i * counter_width),
            (int)Math.round(y + height + 5), (int)Math.round(counter_width + 1),
            (int)Math.round(counter_height + 1));
            rect.adjustFontSize(maxValueString);
        }

    }

}
```