Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А.А. Бабаев, Г.А. Чижова, А.А. Шалыто

Создание скелетной анимации на основе автоматного программирования

Проектная документация

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

Санкт-Петербург
2003

1. Введение

Известны системы для создания скелетной анимации. Они бывают различных типов. Наиболее распространен вариант скелетной анимации (в частности, в играх), в которой кости движутся по жестко заданным траекториям [1]. В настоящее время в основном доступны технологии анимации персонажей на основе уже разработанной скелетной анимации, в то время как подходы к созданию анимации скелетов значительно менее известны.

В настоящей работе описывается решение задачи скелетной анимации, состоящей из двух частей:

- создание скелета, который схематично описывает анимируемую фигуру (статика);
- обеспечение движения фигуры (динамика).

Создание скелета выполняется традиционным путем. При этом предполагается, что скелет состоит из костей, которые связаны между собой в структуру, наиболее естественным образом описывающую анимируемую фигуру. В общем случае эта структура древовидна. Это позволяет автоматически передавать перемещение костей, принадлежащих более высоким уровням иерархии, на кости более низких уровней.

Задача обеспечения движения фигуры, в свою очередь, также состоит из двух частей:

- разработка правил перемещения костей;
- исполнение правил.

В примере скелетной анимации «человека», рассматриваемом в работе, выбраны линейные правила перемещения. Правила содержат подправила, например, правило «перемещение руки» состоит из таких подправил, как «перемещение плечевой кости», «разгибание руки» и т. д.

Особенность предлагаемого подхода состоит в том, что для исполнения этих правил применяется система взаимосвязанных автоматов, которые задаются графами переходов [2].

Это позволяет упростить процедуру программирования скелетной анимации в целом настолько, что если бы авторы не знали автоматного подхода, то его, по их мнению, стоило бы разработать.

В рассматриваемом примере для управления «человеком» применяется три типа автоматов:

- автомат, управляющий перемещением отдельной кости (A0);
- автомат синхронизации фигуры (A1);
- автомат, управляющий работой «мозга» (A2).

Автомат A0, управляющий перемещением кости, получает из «мозга» параметры движения и обеспечивает его выполнение. Этот автомат имеет два состояния:

- «Ожидание»;
- «Перемещение».

Переходы в автомате А0 осуществляются на основе проверки значений следующих входных переменных:

- переменная, инициирующая перемещение;
- переменная, сигнализирующая об окончании перемещения.

Этот автомат выполняет следующие действия:

- инициализирует перемещение;
- выполняет элементарное перемещение кости, вычисляемое как произведение скорости перемещения на время, которое затрачено на данный шаг;
- обеспечивает завершение движения.

В примере используется 15 автоматов данного типа, так как фигура содержит 15 костей.

Автомат синхронизации А1 выделен отдельно, потому что необходимо в одном месте собирать и хранить информацию о факте перемещения костей, а также для обеспечения выполнения следующего действия только после завершения предыдущего. Автомат А1 имеет два состояния:

- «Ожидание выполнения правил»;
- «Выполнение правил».

Переходы в автомате А1 осуществляются на основе проверки значений следующих входных воздействий:

- событие «Уменьшить значение счетчика»;
- событие «Увеличить значения счетчика»;
- переменная, показывающая, что в счетчике осталась единица.

Автомат А1 выполняет следующие действия:

- увеличивает значение в счетчике;
- уменьшает (если это возможно) значение в счетчике.

«Мозг» состоит из двух частей: правил, включающих подправила, и автомата, осуществляющего их выбор, обработку (проверку возможности выполнения) и передачу на выполнение костям.

Автомат А2, управляющий работой мозга, имеет четыре состояния:

- «Ожидание»;
- «Запуск действий»;
- «Ожидание завершения»;
- «Окончание работы».

Переходы в автомате А2 осуществляются на основе проверки значений следующих входных переменных:

- переменная для хранения количества действий в списке правил;
- переменная для хранения количества подправил в текущем правиле;
- переменная, показывающая, что автомат синхронизации А1 находится в состоянии «Выполнение правил».

Автомат А2 выполняет следующие действия:

- удаляет из списка текущее правило, выполняя его;

- убирает из текущего списка подправил первое, передавая его на выполнение кости, для которой оно написано;
- ожидает завершения выполнения всех подправил.

В рамках предлагаемого подхода корректность работы столь большого количества автоматов можно контролировать не только с помощью визуализатора по перемещению фигуры, но и на основе автоматически создаваемых протоколов, которые отображают поведение всех автоматов в системе.

Проект реализован следующим образом. Структура фигуры и правила работы «мозга» задаются в *XML*-формате, и поэтому могут быть легко изменены без перекомпиляции. Остальная часть программы написана на языке *Java*.

Программа содержит 17 классов, из которых все кроме одного, соответствующего костям, порождают по одному объекту. Автоматы в проекте реализованы, как методы класса. Объект («тело») и управляющий им автомат («душа») реализованы совместно.

Рис. 1. Классы, используемые в программе

Программа использует несколько потоков для обеспечения возможности одновременного движения костей. При этом в одном потоке работает «мозг» и синхронизатор, а для каждой из костей создается отдельный поток.

Применение автоматного подхода упрощает внесение изменений в проект, что позволит сравнительно «безболезненно» повысить его функциональность в следующих версиях программы.

2. Постановка задачи

Цель работы состоит в разработке программы, с помощью которой было бы возможно моделировать поведение произвольной системы костей. При этом должны быть решены следующие задачи:

- написание программы визуализации;
- разработка и реализация модели взаимодействия костей;
- разработка и реализация с помощью автоматной технологии программы управления системой;
- исследование возможности и рациональности использования автоматной технологии при разработке данной программы.

По различным причинам в настоящее время реализованы не все поставленные задачи.

Система состоит из «Мозга», «Синхронизатора» и нескольких «Костей» (рис. 2).

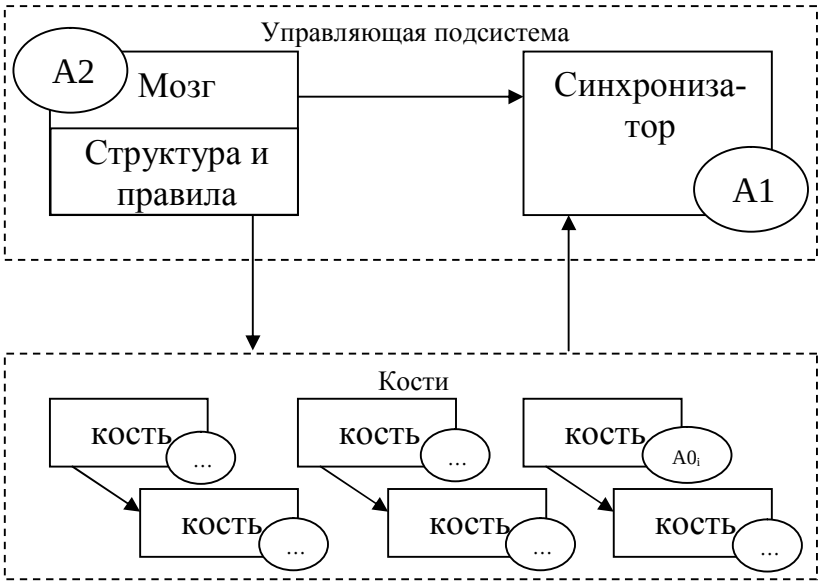

Рис. 2. Структурная схема системы

3. Описание костей

Имеется система «костей» (**Bone**), каждая из которых представляет собой направленный отрезок (выделены начало и конец отрезка) с некоторой массой. При этом кости объединены в древовидную систему таким образом, что каждая кость имеет только одного предка (**Parent**) и ни одного, одного или нескольких потомков (**Child**). Каждая из костей «прикреплена» началом к концу предка. Исключение составляет только кость, которая не имеет предка и называется центральной (**Central Bone**). Кости могут вращаться и перемещаться. При вращении учитываются предельные углы, на которые может поворачиваться данная кость, а при перемещении - длина связей и физические свойства соединения.

При движении одной из костей, остальные перемещаются согласно следующим законам:

- центральная кость не может вращаться;
- потомки перемещаемой кости перемещаются так же, как и предок;
- предок перемещаемой кости вращается для того, чтобы перемещаемая кость смогла бы переместиться в нужную точку. Если этого недостаточно, то предок «пытается» переместиться в такую точку, чтобы перемещаемая кость смогла завершить движение. Если и это невозможно, то движение данной кости завершается.

4. Описание физических свойств костей

Свойства кости:

- масса. Считается, что кость является цилиндром и масса распределена равномерно. Центр масс находится в геометрическом центре цилиндра;
- длина. Она постоянна и не зависит от каких-либо параметров.

Свойства соединения кости:

- жесткость – это свойство определяет насколько жестко прикреплена одна кость к другой. Чем больше эта величина, тем более хорошо передается движение от предка к потомку. Допустим, что у нас есть две кости и жесткость соединения между ними равна X. Если эта величина равна нулю, то соединение не жесткое, и как не вертеть предка, потомок останется в исходном положении;
- инерционность – это свойство показывает насколько кость стремится сохранить свое состояние, независимо от того, находится ли она в неподвижном состоянии или в движении.

5. Правила перемещения костей

Эти правила являются упрощенным решением физической задачи произвольного перемещения двух нерастяжимых стержней, связанных между собой.

1. Потомок может вращаться только относительно точки, в которой он в данный момент крепится к предку.
2. При повороте кости, все остальные, прикрепленные к нему двигаются вместе с ней, как одно целое.
3. Перемещение кости возможно только в случае, если это центральная кость.

В данный момент реализовано перемещение всей фигуры с помощью перемещения центральной кости и движение фигуры за счет поворота отдельных костей.

6. Описание входных данных

В программе реализована работа с обобщенной моделью проволочной системы костей. Для создания конкретной модели необходимо написать файл

описания системы и файл описания действий. Далее приводятся форматы файлов описаний.

6.1. Файл описания системы костей

Файл описания системы костей содержит информацию для построения конкретной модели. В нем с помощью языка разметки документов *XML* описывается информация о модели в следующем формате.

Описание фигуры:

```
<figure name="Имя модели" author="Авторы"
   date="Дата последнего изменения модели">
    описание кости
        …
    описание кости
</figure>
```

Описание каждой кости:

```
<Bone name="Имя кости" parent="Имя родителя
   кости" mass="Масса" length="Длина">
    <location x="x" y="y" z="z" /> - Положение
          центра масс.
    <angles x="x" y="y" z="z" /> - Углы поворота,
          относительно кости родителя.
    <minAngles x="x" y="y" z="z" /> - Минимально
          допустимый угол поворота, относительно кости
          родителя.
    <maxAngles x="x" y="y" z="z" /> - Максимально
          допустимый угол поворота, относительно кости
          родителя.
</Bone>
```

Имя кости может быть любым, однако выделено несколько стандартных имен, которыми можно пользоваться при описании:

- **`null`** – нет имени;
- **`centralBone`** – центральная кость;
- **`body`** – тело;
- **`head`** – голова;
- **`leftHand (rightHand)`** – левая (правая) рука;
- **`leftLeg (rightLeg)`** – левая (правая) нога.

Формат файла описания системы костей приведена в приложении 1. Пример соответствующего файла для рассматриваемой системы скелетной анимации приведен в приложении 2.

6.2. Файл описания движений костей

Действия, которые будут происходить с костями, описаны в файле на языке разметки документов *XML*.

1. Описание действий:

```
<actions name="Имя действия" author="Авторы"
   date="Дата создания">
    изменение позиции
        …
    изменение позиции
</actions>
```

2. Описание изменения позиции:

```
<position name="Имя позиции">
    изменение положения кости
        …
    изменение положения кости
</position>
```

3. Описание изменения положения кости:

```
<BoneAction name="Имя кости"
   speedTranslate="Скорость поступательного
   движения" speedRotate="Скорость вращения">
     <angles x="x" y="y" z="z" /> -
          Поворот кости относительно кости родителя.
     <location x="x" y="y" z="z" /> -
          Поступательной перемещение кости в пространстве
          относительно текущего положения.
</BoneAction>
```

При совершении описанного действия над костью с заданным именем в системе также будет автоматически изменяться положение всех связанных с ней костей.

4. Последовательность производимых действий:

```
<action type="тип действия" value="Имя
   описанного действия" />
```

При чтении файла создается список действий, последовательность которых описана внутри блоков **<action… />**. Сами действия – описаны в блоках **<position… />**.

Формат файла описания движения костей приведена в приложении 3. Пример соответствующего файла для рассматриваемой системы скелетной анимации приведен в приложении 4.

6.3. Протоколирование

Система протоколирования основана на стандартном *Java*-классе **«Logger»**. Это позволяет протоколировать события как на экране, так и в файле, причем, в произвольном, определяемом пользователем формате.

Поскольку программа многопотоковая, она постоянно запускает автоматы управления костями. При этом, чтобы уменьшить размер файла протокола, пришлось запретить вызовы автоматов в случае, если заведомо никаких действий не произойдет.

Пример фрагмента протокола приведен в приложении 5.

7. Система управления

7.1. Класс «Кость» («**Bone**»)

7.1.1. Словесное описание

Класс содержит информацию о физических параметрах кости, его родителе и текущем положении. Автомат A0 отслеживает перемещение кости, следит за окончанием перемещения.

7.1.2. Входные переменные

- **x0_0** – индикатор необходимости перемещения. Принимает значение «1», когда перемещение необходимо (выполняется какое-либо действие), «0» – если никакое перемещение не требуется;
- **x0_1** – индикатор окончания перемещения. Принимает значение «1», когда перемещение закончено, «0» – если кость еще не дошла до конечного состояния.

7.1.3. Выходные воздействия

- **z0_0** – действия необходимые для того, чтобы в функции **z0_1** корректно рассчитать координаты (инициализация перемещения);
- **z0_1** – передвинуть кость на *dx*, где *dx* – элементарное перемещение, вычисляемое, как произведение скорости перемещения и времени, которое затрачено на данный шаг;

- **z0_2** – действия для окончания перемещения, в частности, посылка автомату **A1** события e1.

7.1.4. Схема связей автомата

Схема связей этого автомата из-за малого числа входных и выходных воздействий в работе не приводится.

7.1.5. Граф переходов

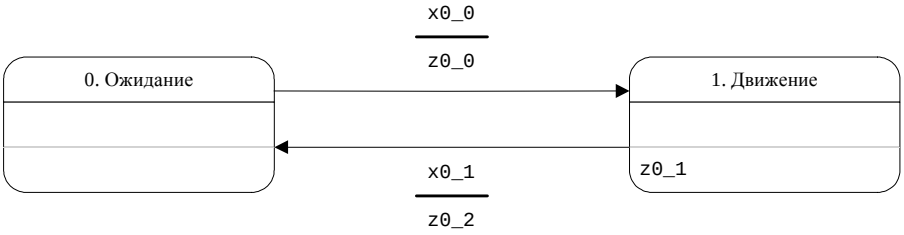

Рис. 3. Граф переходов автомата A0

7.2. Класс «Синхронизатор» («Synchronizer»)

7.2.1. Словесное описание

Класс содержит информацию о текущем выполнении действия. Точнее, он содержит число, которое показывает, сколько поддействий текущего действия еще выполняется. Как только это число становится равным нулю, можно начинать выполнять следующее действие. Автомат A1 запускается при передаче ему события «Увеличить значение счетчика» и «Уменьшить значение счетчика».

7.2.2. Входные воздействия

- **e0** – увеличить значение счетчика;
- **e1** – уменьшить значение счетчика;
- **x1_1** – в счетчике осталась единица.

7.2.3. Выходные воздействия

- **z1_0** – увеличить значение счетчика;
- **z1_1** – уменьшить (если это возможно) значение счетчика.

7.2.4. Схема связей автомата

Схема связей этого автомата из-за малого числа входных и выходных воздействий в работе не приводится.

7.2.5. Граф переходов

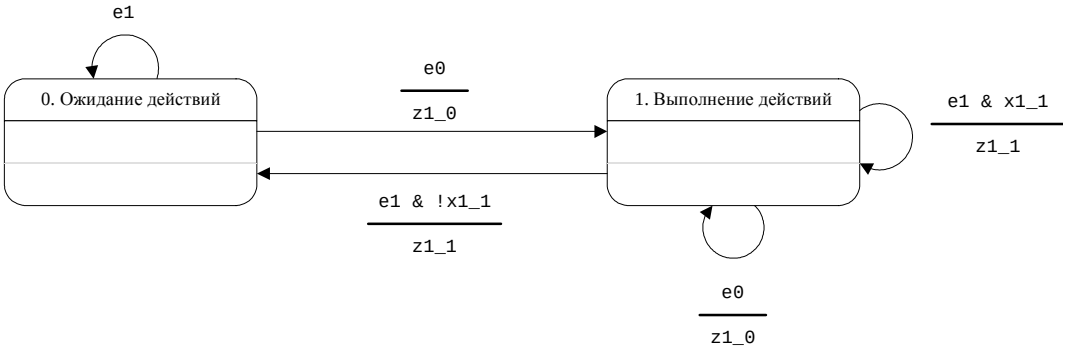

Рис. 4. Граф переходов автомата A1

7.3. Класс «Мозг» («**Actions**»)

7.3.1. Словесное описание

В файле описания действий записаны правила, каждое из которых делится на подправила. Правила определяют действия (перемещения) скелета в целом. Подправила задают движение отдельных костей. При этом поведение каждой кости может описываться отдельным подправилом.

Автомат A2 последовательно выполняет правила, передавая на выполнение каждой кости соответствующее подправило. После этого автомат переходит к выполнению следующего правила. В рассматриваемом примере (приложение 4) для управления скелетом «в мозге» находится семь правил, которые содержат по одному или по два подправила. Благодаря использованию языка *XML* обеспечиваются гибкость описания и простота изменения поведения скелета.

7.3.2. Входные переменные

- **x2_0** – количество правил;
- **x2_1** – количество подправил в текущем правиле.

7.3.3. Выходные воздействия

- **z2_0** – выполнить текущее правило;
- **z2_1** – передать на выполнение текущее подправило;
- **z2_2** – ожидать завершения выполнения всех подправил текущего правила.

7.3.4. Схема связей автомата

Схема связей этого автомата из-за малого числа входных и выходных воздействий в работе не приводится.

7.3.5. Граф переходов

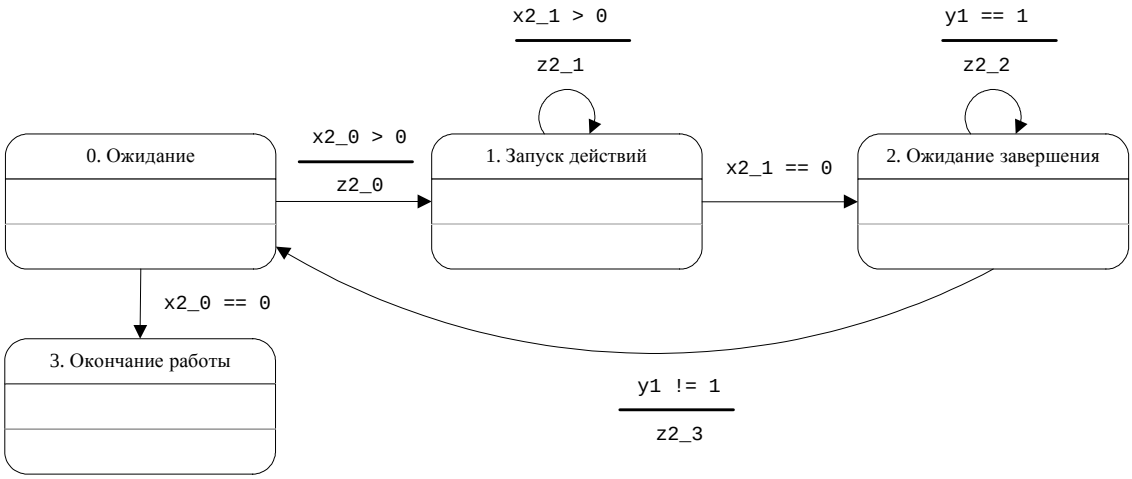

Рис. 5. Граф переходов автомата A2

8. Сравнение традиционного объектно-ориентированного программирования и ООП с явным выделением состояний

Для рассматриваемой задачи было написано две программы. В первой был использован традиционный подход к объектно-ориентированному программированию (ООП), а во второй – ООП с явным выделением состояний [2].

После такой двойной работы над одним и тем же приложением можно определить достоинства и недостатки каждого из подходов.

Традиционный объектный подход сегодня стандартен и привычен для многих. Это его несомненное достоинство. Другое достоинство состоит в том, что текст, написанный квалифицированным программистом можно «прочитать».

Однако он не лишен недостатков, главный из которых состоит в том, что сложную программу весьма трудно понять. Это напоминает ситуацию, когда человек может читать на иностранном языке, но понимает «через слово».

Именно потому, что текст можно «прочитать», большинство программистов не пишет проектную документацию. Обычно в качестве документации выступают самодокументирующиеся программы. При этом часто в одном файле используются одновременно русский (комментарии) и английский (текст программы) языки. В настоящее время наибольшее достижение состоит в получении программной документации по выделенным кускам программного текста, например, комментариям. Такая документация создается с помощью системы *JavaDoc*.

Это конечно не плохо, но это не является проектной документацией, по которой можно разобраться со структурой программы и её поведением. Для этой цели в настоящее время разработан язык *UML* [3]. Однако при сложном

поведении системы, по мнению авторов работы [4], использование этого языка практически невозможно. Как, например, построить диаграммы взаимодействия объектов в начале программирования?

При использовании ООП с явным выделением состояний, предложенного в работе [4], выпуск проектной документации становится неизбежным.

Для сравнения подходов рассмотрим два фрагмента программы, делающих одно и то же. Сначала приведем фрагмент программы без использования автоматов (листинг 1).

Листинг 1. Управление костью без использования автоматов

```
state = ST_WAITING;

while (true)
{
  try
  {
    yield();
    sleep(delay);
  }
  catch (InterruptedException e)
  {
    e.printStackTrace();
  }
  switch (state)
  {
    case (ST_WAITING) :
    {
       if (curLoc.x != loc.x ||
           curLoc.y != loc.y ||
           curLoc.z != loc.z ||
           curAngles.x != angles.x ||
           curAngles.y != angles.y ||
           curAngles.z != angles.z)
       {
          time = new Date().getTime();
          state = ST_TURNING;
       }
       break;
    }
    case (ST_TURNING) :
    {
      checkAndIncrement();

      if (state == ST_WAITING)
        signaller.removeSignal();

      update();

      break;
    }
  }
}
```

Такой код, даже не зная назначения вызываемых функций, может прочитать человек, который знаком с используемым языком программирования. Понять его сложнее – требуется большая квалификация, чем только для чтения.

Вносить же изменения в так написанный текст – весьма опасное занятие, так как программа может полностью «развалиться» и просто перестать работать.

Теперь приведем пример того же кода, переписанного с использованием автоматного подхода (листинг 2):

Листинг 2. Управления костью с использованием автомата

```
public void run()
{
    while (true)
    {
        try
        {
            yield();
            sleep(delay);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        A0();
    }
}

void A0()
{
    switch (y0)
    {
        case 0:
            if (x0_0()) { z0_0();               y0 = 1; }
        break;
        case 1:
            z0_1();
            if (x0_1()) { z0_2();               y0 = 2; }
        break;
    }
}
```

Такой код тоже легко читается, но его на первый взгляд трудно понять, так как используются не смысловые, а символьные идентификаторы. Однако при использовании автоматного подхода код программы не является единственным документом для ее понимания. В ООП с явным выделением состояний в общем случае предлагается использовать следующие документы:

- диаграмма классов;
- структурная схема каждого класса;
- схема взаимодействий автоматов;
- для каждого автомата:
  - словесное описание, которое является «декларацией о намерениях»;
  - схема связей автоматов, в которой описываются входные и выходные воздействия на русском языке;
  - граф переходов с символьными обозначениями входных и выходных воздействий и названиями состояний, записанными по-русски;

    - текст программы, построенный по графу переходов формально и изоморфно с символьными обозначениями, в которых смысловые идентификаторы и комментарии отсутствуют;
- автоматически получаемые протоколы взаимодействия компонент системы в терминах объектов, автоматов, состояний, входных и выходных воздействий, а не переменных и названий функций, как это делается традиционно.

Отметим также, что при использовании традиционного ООП обычно логика объектного кода часто теряется за технологическими особенностями – конструкциями конкретного языка программирования. Так, например, в листинге 1, гораздо сложнее, чем в листинге 2, понять, куда и почему передается управление. Автоматный подход обеспечивает отделение логики от технических частей программы, что делает его в определенном смысле понятнее.

9. Заключение

Данная работа не претендует на полноту. В ней были реализованы идеи, которые показались нам интересными в контексте автоматного программирования.

При этом мы думаем, что затронули довольно объемную, и интересную для работы тему.

В результате была разработана программа, которую можно считать основой для разработки более сложных систем управления тем, что у нас названо системой костей. В реальной жизни это несложные роботы, при моделировании которых можно пользоваться данной моделью. Степень доработки определяется, главным образом тем, что в нашей модели все кости имеют одинаковые параметры и двигаются по одинаковым законам, а, на самом деле, голова может приводиться в движение совершенно иначе, чем рука скелета. При этом придется писать различные автоматы для различных частей.

10. Пример работающего приложения

На рис. 6 показано работающее приложение. Зеленым цветом показаны концы костей, а красным – их начало. Синие точки – центры масс. Размеры точек соответствуют массам костей. Система костей описана в приложении 2, а движения костей – в приложении 4.

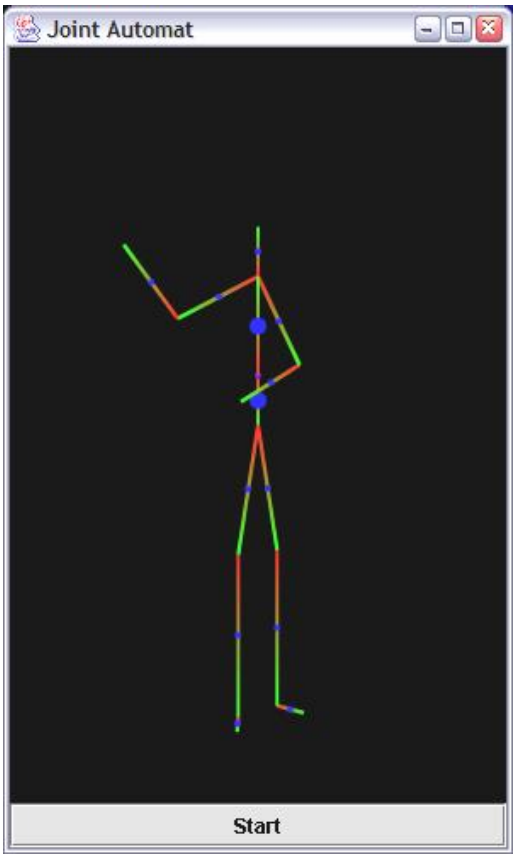

Рис. 6. Пример работающего приложения

11. Исходные тексты классов

Исходные тексты и скомпилированные классы находятся в прилагаемых архивах и в приложении 6.

Для того, чтобы запустить программу, требуется среда исполнения *Java Runtime Engine* версии 1.4 или старше и библиотека *Java 3D* версии 1.3 или старше. Скачать их можно на сайте `http://java.sun.com`.

Источники

1. *Ваткин С.* Скелетная анимация. http://www.gamedev.ru/coding/10917.shtml.
2. *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5. http://is.ifmo.ru, раздел «Статьи».
3. http://www.uml.org/
4. *Шалыто А.А., Туккель Н.И.* Танки и автоматы //BYTE/Россия. 2003. № 1. http://is.ifmo.ru, раздел «Статьи».

Приложения

1. Формат файла описания системы костей (**figure.dtd**)

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

<!— Описание корневого элемента XML-файла -->
<!ELEMENT figure (Bone+)>
    <!ATTLIST automat name CDATA #REQUIRED
                      author CDATA #IMPLIED
                      date CDATA #IMPLIED>

<!-- Элемент описания кости -->
<!ELEMENT Bone (location?, angles, minAngles, maxAngles, massCentre?)>
    <!ATTLIST Bone name CDATA #REQUIRED
                   parent CDATA #REQUIRED
                   mass CDATA #REQUIRED
                   length CDATA #REQUIRED>

<!— Элемент описания положения кости -->
<!ELEMENT location (CDATA)>
    <!ATTLIST location x CDATA #REQUIRED
                       y CDATA #REQUIRED
                       z CDATA #REQUIRED>

<!— Элемент описания углов расположения кости -->
<!ELEMENT angles (CDATA)>
    <!ATTLIST angles x CDATA #REQUIRED
                     y CDATA #REQUIRED
                     z CDATA #REQUIRED>

<!— Элемент описания минимально возможных углов для кости -->
<!ELEMENT minAngles (CDATA)>
    <!ATTLIST minAngles x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        z CDATA #REQUIRED>

<!-- Элемент описания максимально возможных углов для кости -->
<!ELEMENT maxAngles (CDATA)>
    <!ATTLIST maxAngles x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        z CDATA #REQUIRED>

<!-- Элемент описания положения центра масс кости -->
<!ELEMENT massCentre (CDATA)>
    <!ATTLIST massCentre x CDATA #REQUIRED
                         y CDATA #REQUIRED
                         z CDATA #REQUIRED>
```

2. Пример файла описания системы костей

```
<?xml version="1.0"?>

<figure name="Test Figure" author="Galina Chizhova, Alexander Babaev"
date="12.11.2002">

<!— Описание центральной кости -->
<Bone name="centralBone" parent="null" mass="0" length="0">
  <location x="0" y="0" z="10" />
  <angles x="0" y="0" z="0" />
  <minAngles x="0" y="0" z="0" />
  <maxAngles x="0" y="0" z="0" />
</Bone>

<!— Описание «грудины» -->
<Bone name="body" parent="centralBone" mass="20" length="10">
  <angles x="0" y="-180" z="0" />
  <minAngles x="0" y="0" z="-30" />
  <maxAngles x="0" y="0" z="30" />
</Bone>

<!— Описание «головы» -->
<Bone name="head" parent="body" mass="5" length="5">
  <angles x="0" y="0" z="40" />
  <minAngles x="0" y="180" z="0" />
  <maxAngles x="0" y="180" z="0" />
</Bone>

<!— Описание «таза» -->
<Bone name="pelvis" parent="centralBone" mass="20" length="5">
  <angles x="0" y="0" z="0" />
  <minAngles x="0" y="0" z="-30" />
  <maxAngles x="0" y="0" z="30" />
</Bone>

<!— Описание «левого плеча» -->
<Bone name="leftShoulder" parent="body" mass="4" length="10">
  <angles x="0" y="120" z="0" />
  <minAngles x="0" y="-180" z="-90" />
  <maxAngles x="180" y="180" z="90" />
</Bone>

<!— Описание «левого предплечья» -->
<Bone name="leftHand" parent="leftShoulder" mass="4" length="8">
  <angles x="0" y="-80" z="0" />
  <minAngles x="-180" y="-180" z="-90" />
  <maxAngles x="0" y="180" z="90" />
</Bone>

<!— Описание «правого плеча» -->
<Bone name="rightShoulder" parent="body" mass="4" length="10">
  <angles x="0" y="-150" z="0" />
  <minAngles x="0" y="-180" z="-90" />
  <maxAngles x="180" y="180" z="90" />
</Bone>
```

```
<!— Описание «правого предплечья» -->
<Bone name="rightHand" parent="rightShoulder" mass="4" length="8">
  <angles x="0" y="-90" z="0" />
  <minAngles x="0" y="-180" z="-90" />
  <maxAngles x="180" y="180" z="90" />
</Bone>

<!— Описание «левого бедра» -->
<Bone name="leftThigh" parent="pelvis" mass="7" length="13">
  <angles x="0" y="-10" z="0" />
  <minAngles x="-90" y="-80" z="-70" />
  <maxAngles x="10" y="30" z="70" />
</Bone>

<!— Описание «левой голени» -->
<Bone name="leftLeg" parent="leftThigh" mass="7" length="16">
  <angles x="0" y="10" z="0" />
  <minAngles x="-10" y="-30" z="-70" />
  <maxAngles x="90" y="80" z="70" />
</Bone>

<!— Описание «левой стопы» -->
<Bone name="leftFoot" parent="leftLeg" mass="7" length="3">
  <angles x="0" y="-90" z="-60" />
  <minAngles x="-10" y="-30" z="-70" />
  <maxAngles x="90" y="80" z="70" />
</Bone>

<!— Описание «правого бедра» -->
<Bone name="rightThigh" parent="pelvis" mass="7" length="13">
  <angles x="0" y="10" z="0" />
  <minAngles x="-10" y="-30" z="-70" />
  <maxAngles x="90" y="80" z="70" />
</Bone>

<!— Описание «правой голени» -->
<Bone name="rightLeg" parent="rightThigh" mass="7" length="16">
  <angles x="0" y="-10" z="0" />
  <minAngles x="-10" y="-30" z="-70" />
  <maxAngles x="90" y="80" z="70" />
</Bone>

<!— Описание «правой стопы» -->
<Bone name="rightFoot" parent="rightLeg" mass="7" length="3">
  <angles x="0" y="90" z="60" />
  <minAngles x="-10" y="-30" z="-70" />
  <maxAngles x="90" y="80" z="70" />
</Bone>
</figure>
```

3. Формат файла описания движения костей (**actions.dtd**)

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

<!— Описание корневого элемента XML-файла -->
<!ELEMENT actions (position+, action+)>
    <!ATTLIST actions name CDATA #REQUIRED
                      author CDATA #IMPLIED
                      date CDATA #IMPLIED>


<!-- Элемент описания позиции системы -->
<!ELEMENT position (BoneAction+)>
    <!ATTLIST position name CDATA #REQUIRED>

<!-- Элемент описания позиции отдельной кости -->
<!ELEMENT BoneAction (location?, angles?)>
    <!ATTLIST Bone name CDATA #REQUIRED
                    speedTranslate CDATA
                    speedRotate CDATA>

<!-- Элемент описания позиции кости -->
<!ELEMENT location (CDATA)>
    <!ATTLIST location x CDATA #REQUIRED
                        y CDATA #REQUIRED
                        z CDATA #REQUIRED>

<!-- Элемент описания углов перемещения кости -->
<!ELEMENT angles (CDATA)>
    <!ATTLIST angles x CDATA #REQUIRED
                     y CDATA #REQUIRED
                     z CDATA #REQUIRED>

<!-- Элемент описания правила -->
<!ELEMENT action (CDATA)>
    <!ATTLIST action type CDATA #REQUIRED
                     value CDATA #REQUIRED>
```

4. Пример файла описания движения костей

```
<?xml version="1.0"?>
<actions name="Step" author="Alexander Babaev" date="12.11.2002">

<!-- Действие «Поднять правую ногу» -->
<position name="rightLegUp">
  <!-- Подправило 1 -->
  <BoneAction name="rightThigh" speedRotate="0.1">
    <angles x="-10" y="10" z="0" />
  </BoneAction>

  <!-- Подправило 2 -->
  <BoneAction name="rightLeg" speedRotate="0.05">
    <angles x="10" y="-10" z="0" />
  </BoneAction>
</position>
```

```
<!-- Действие «Опустить правую ногу» -->
<position name="rightLegDown">
  <BoneAction name="rightThigh" speedRotate="0.1">
    <angles x="0" y="10" z="0" />
  </BoneAction>

  <BoneAction name="rightLeg" speedRotate="0.05">
    <angles x="0" y="-10" z="0" />
  </BoneAction>
</position>

<!-- Действие «Поднять левую ногу» -->
<position name="leftLegUp">
  <BoneAction name="leftThigh" speedRotate="0.1">
    <angles x="-10" y="-10" z="0" />
  </BoneAction>

  <BoneAction name="leftLeg" speedRotate="0.05">
    <angles x="10" y="10" z="0" />
  </BoneAction>
</position>

<!-- Действие «Опустить левую ногу» -->
<position name="leftLegDown">
  <BoneAction name="leftThigh" speedRotate="0.1">
    <angles x="0" y="-10" z="0" />
  </BoneAction>

  <BoneAction name="leftLeg" speedRotate="0.05">
    <angles x="0" y="10" z="0" />
  </BoneAction>
</position>



<!-- Описание последовательности вызовов правил -->

<!-- Перемещение кости (два подправила) -->
<action type="position" value="rightLegUp" />

<!-- Временная задержка (одно подправило) -->
<action type="delay" value="100" />

<!-- Перемещение кости (два подправила) -->
<action type="position" value="rightLegDown" />

<!-- Временная задержка (одно подправило) -->
<action type="delay" value="100000" />

<!-- Перемещение кости (два подправила) -->
<action type="position" value="leftLegUp" />

<!-- Временная задержка (одно подправило) -->
<action type="delay" value="100" />

<!-- Перемещение кости (два подправила) -->
<action type="position" value="leftLegDown" />

</actions>
```

5. Фрагмент протокола

```
>>> A2 (Actions "Step")
    A2->z2_0 ("Step" action processing started)
    A2->z2_1 ("Step" action processing...)
 +  A1->z1_0 (add a signal)
 +  A1->z1_0 (add a signal)
    A2->z2_2 ("Step" waiting for action to finish...)
    A2->z2_2 ("Step" waiting for action to finish...)
>>> A0 (Bone "rightLeg")
    A2->z2_2 ("Step" waiting for action to finish...)
    A0->z2_1 (start moving Bone "rightLeg" ...)
    A2->z2_2 ("Step" waiting for action to finish...)
    A2->z2_2 ("Step" waiting for action to finish...)
 %  A0->z2_1 (moving a bit Bone "rightLeg" ...)
    A2->z2_2 ("Step" waiting for action to finish...)
      …
    A2->z2_2 ("Step" waiting for action to finish...)
>>> A0 (Bone "rightThigh")
    A0->z2_1 (start moving Bone "rightThigh" ...)
 %  A0->z2_1 (moving a bit Bone "rightThigh" ...)
    A2->z2_2 ("Step" waiting for action to finish...)
      …
    A2->z2_2 ("Step" waiting for action to finish...)
 %  A0->z2_1 (moving a bit Bone "rightThigh" ...)
    A2->z2_2 ("Step" waiting for action to finish...)
      …
    A2->z2_2 ("Step" waiting for action to finish...)
 %  A0->z2_1 (moving a bit Bone "rightLeg" ...)
    A2->z2_2 ("Step" waiting for action to finish...)
      …
```

6. Исходный текст программы на языке *Java*

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\Main.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: 04.11.2002
 * Time: 13:03:11
 */
package ru.ezhiki.automat.human;
import ru.ezhiki.automat.human.vis.AutomatFrame;
import ru.ezhiki.automat.human.logic.*;
import ru.ezhiki.utils.AutomatSimpleFormatter;
import java.util.logging.Logger;
import java.util.logging.StreamHandler;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
public class Main
{
   public static void main(String[] args)
   {
      Logger logger = Logger.getLogger("ru.ezhiki.automat.human.logger");
      try
      {
         logger.addHandler(new StreamHandler(new FileOutputStream("automat.log"),
             new AutomatSimpleFormatter()));
      }
      catch (SecurityException e)
      {
         e.printStackTrace();
      }
      catch (FileNotFoundException e)
      {
         e.printStackTrace();
      }
      Actions actions = new ActionsLoader().loadFile("samples/actionsSample.xml");
      Figure figure = new FigureLoader().loadFile("samples/figureSample.xml");
      AutomatFrame frame = new AutomatFrame(figure, actions, logger);
      frame.show();
   }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\MainApplet.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 11:34:51 PM
 */
package ru.ezhiki.automat.human;
import ru.ezhiki.automat.human.logic.*;
import ru.ezhiki.automat.human.vis.AutomatFrame;
```

```
import java.applet.Applet;
import java.util.logging.Logger;
public class MainApplet extends Applet
{
   private Figure figure = null;
   private Actions actions = null;
```

```
   public MainApplet()
   {
   }
```

```
   public void init()
   {
      super.init();
      figure  = new FigureLoader().loadFile ("figureSample.xml");
      actions = new ActionsLoader().loadFile("actionsSample.xml");
   }
```

```
   public void start()
   {
      super.start();
      Logger logger = Logger.getLogger("ru.ezhiki.automat.human.logger");
      AutomatFrame frame = new AutomatFrame(figure, actions, logger);
      frame.show();
   }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\Action.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 9:55:23 PM
 */
package ru.ezhiki.automat.human.logic;
/**
 * Action, that can be done by figure. Actions can be only two types: delay and repositioning of joints.
 */
public class Action
{
   /**
    * Type of action. Can be any string, but currently supported are 'delay' and 'position'
    */
   private String type = "";
   /**
    * Value for this action. Each action interprets this variable, as it needs to.
    */
   private String value = "";
```

```
   /**
    * Constructor.
    */
   public Action(String type, String value)
   {
      this.type = type;
      this.value = value;
   }
```

```
    /**
     * Returns type of action.
     */
    public String getType()
    {
        return type;
    }
```

```
    /**
     * Returns value for this action.
     */
    public String getValue()
    {
        return value;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\Actions.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 9:44:39 PM
 */
package ru.ezhiki.automat.human.logic;

import ru.ezhiki.utils.Debug;
import ru.ezhiki.utils.Signaller;

import java.util.*;
import java.util.logging.Logger;

public class Actions extends Thread
{
    private String name = "No Name";
    private String authors = "Unknown";
    private String date = "Unknown";

    public HashMap positions = null;
    public Vector actions = null;

    private Figure figure = null;

    private Signaller signaller = Signaller.getSignaller();
    public static long waitingDelay = 1;

    private int y0 = 0;
    private int curAction = 0;
    private int curSubAction = 0;
    private Logger logger = null;
```

```
    public Actions(String name, String authors, String date)
    {
        this.name = name;
        this.authors = authors;
        this.date = date;
```

```
    actions = new Vector();
    positions = new HashMap();
}
```

```
public String getActionsName()
{
    return name;
}
```

```
public String getAuthors()
{
    return authors;
}
```

```
public String getDate()
{
    return date;
}
```

```
public void addPosition(Position aPosition)
{
    positions.put(aPosition.getName(), aPosition);
}
```

```
public void addAction(Action action)
{
    actions.add(action);
}
```

```
public void setFigure(Figure figure)
{
    this.figure = figure;
}
```

```
public void processActions()
{
    curAction = 0;
    curSubAction = 0;
    y0 = 0;
    A2();
}
```

```
private void A2()
{
    if (logger != null)
        logger.info(">>> A2 (Мозг начал работу (автомат \"" + name +  "\"))");
```

```
   while (y0 != 3)
   {
      switch (y0)
      {
         case 0:
            if (x2_0() > 0)      {z2_0();                y0 = 1; }
            else
            if (x2_0() == 0)     {                       y0 = 3; }
         break;
         case 1:
            if (x2_1() > 0)      {z2_1();                      }
            else
            if (x2_1() == 0)     {                       y0 = 2; }
         break;
         case 2:
            if (x2_2() == true)  {z2_2();                      }
            else
            if (x2_2() == false) {z2_3();                y0 = 0; }
         break;
      }
   }

   if (logger != null)
      logger.info("<<< A2 (Мозг завершил работу (автомат \"" + name +  "\"))");
}
```

```
private int x2_0()
{
   return actions.size() - curAction;
}
```

```
private int x2_1()
{
   String type = ((Action)actions.elementAt(curAction)).getType();
   if (type.equals("delay"))
      return 1 - curSubAction;
   else if (type.equals("position"))
   {
      return 1 - curSubAction;
   }
   else
      return 0;
}
```

```
private boolean x2_2()
{
   synchronized (signaller)
   {
      return signaller.hasSignals();
   }
}
```

```
private void z2_0()
{
```

```
    if (logger != null)
      logger.info("    A2->z2_0 (Начато выполнение действия: (автомат \"" + name +  "\"))");

    curSubAction = 0;
  }
```

```
  private void z2_1()
  {
    if (logger != null)
      logger.info("    A2->z2_1 (Выполняется действие: (автомат \"" + name +  "\"))");

    Action action = (Action)actions.elementAt(curAction);
    String type = ((Action)actions.elementAt(curAction)).getType();
    if (type.equals("delay"))
    {
      try
      {
        sleep(new Long(action.getValue()).longValue());
      }
      catch (InterruptedException e)
      {
        e.printStackTrace();
      }
      catch (NumberFormatException e)
      {
        e.printStackTrace();
      }
    }
    else if (type.equals("position"))
    {
      Position newPos = (Position)positions.get(action.getValue());
      Set nameSet = newPos.getJointNames();

      synchronized (signaller)
      {
        signaller.clear();
      }

      for (Iterator iterator = nameSet.iterator(); iterator.hasNext();)
      {
        JointAction jAction = newPos.getJointAction((String)iterator.next());
        Joint joint = figure.getJointByName(jAction.getName());

        if (joint != null)
        {
          joint.setSpeeds(jAction.getSpeedTurn(), jAction.getSpeedTranslate());
          joint.setAnglesAndLocation(jAction.getAngles(), jAction.getLocation());
          synchronized (signaller)
          {
            signaller.addSignal();
          }
        }
        else
        {
          if (Debug.isDebug())
            System.out.println("Cant't find joint with name :'" + jAction.getName() + "'");
        }
      }
    }

    curSubAction++;
```

```
    }
```

```
    private void z2_2()
    {
        if (logger != null)
            logger.info("    A2->z2_2 (Ожидание завершения действия (автомат \"" + name +  "\"))");

        synchronized (signaller)
        {
            if (Debug.isDebug())
                System.out.println("number of working joints: " + signaller.toString().length());

            try
            {
                yield();
                sleep(waitingDelay);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
```

```
    private void z2_3()
    {
        if (logger != null)
            logger.info("    A2->z2_3 (Поиск следующего действия... (автомат \"" + name +  "\"))");
        curAction++;
    }
```

```
    public void setLogger(Logger logger)
    {
        this.logger = logger;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\ActionsLoader.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 9:42:18 PM
 */
package ru.ezhiki.automat.human.logic;

import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.*;

import java.io.FileReader;

import ru.ezhiki.utils.MathUtils;
```

```
/**
 * Loads actions from an xml-file.
 */
public class ActionsLoader extends DefaultHandler
{
   /**
    * Storage of loaded actions.
    */
   private Actions actions = null;

   private Position currentPosition = null;
   private Action currentAction = null;
   private JointAction currentJointAction = null;
```

```
   /**
    * Look the overrided method description.
    */
   public void startElement(String uri, String localName,
                            String qName, Attributes attr)
         throws SAXException
   {
      if (localName.equals("actions"))
      {
         actions = new Actions(attr.getValue("name"), attr.getValue("authors"), attr.getValue("date"));
      }
      else if (localName.equals("position"))
      {
         currentPosition = new Position(attr.getValue("name"));
      }
      else if (localName.equals("jointAction"))
      {
         currentJointAction = new JointAction(attr.getValue("name"), attr.getValue("speedTurn"),
attr.getValue("speedTranslate"));
      }
      else if (localName.equals("location"))
      {
         currentJointAction.setLocation(new Double(attr.getValue("x")).doubleValue(),
                                        new Double(attr.getValue("y")).doubleValue(),
                                        new Double(attr.getValue("z")).doubleValue());
      }
      else if (localName.equals("angles"))
      {
         currentJointAction.setAngles(MathUtils.deg2rad(new Double(attr.getValue("x")).doubleValue()),
                                      MathUtils.deg2rad(new Double(attr.getValue("y")).doubleValue()),
                                      MathUtils.deg2rad(new Double(attr.getValue("z")).doubleValue()));
      }
      else if (localName.equals("action"))
      {
         currentAction = new Action(attr.getValue("type"), attr.getValue("value"));
      }
   }
```

```
   /**
    * Look the overrided method description.
    */
   public void endElement(String uri, String localName, String qName)
         throws SAXException
   {
```

```
        if (localName.equals("actions"))
        {
        }
        else if (localName.equals("position"))
        {
            actions.addPosition(currentPosition);
        }
        else if (localName.equals("jointAction"))
        {
            currentPosition.addJointAction(currentJointAction);
        }
        else if (localName.equals("action"))
        {
            actions.addAction(currentAction);
        }
    }
```

```
    /**
     * Constructor.
     */
    public ActionsLoader()
    {
    }
```

```
    /**
     * Loads the file and returns actions object.
     */
    public Actions loadFile(String fName)
    {
        try
        {
            XMLReader reader = XMLReaderFac-
tory.createXMLReader("org.apache.crimson.parser.XMLReaderImpl");
            reader.setContentHandler(this);
            reader.parse(new InputSource(new FileReader(fName)));
        }
        catch (SAXParseException e)
        {
            System.out.println("Error at line: " + e.getLineNumber() + ", column: " + e.getColumnNumber() + ",
Id: " + e.getPublicId());
            e.printStackTrace();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        return actions;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\Figure.java

```
/*
 * Created by IntelliJ IDEA.
 * User: chizhik
```

```
 * Date: 03.11.2002
 * Time: 11:20:31
 */
package ru.ezhiki.automat.human.logic;

import ru.ezhiki.automat.human.logic.Joint;

import java.util.Vector;
import java.util.HashMap;
import java.util.Iterator;
import java.util.logging.Logger;


/**
 * Holder for joints.
 */
public class Figure extends Thread
{
    /**
     * Name of the figure. Can be any string.
     */
    private String name = "No Name";
    /**
     * Author of the figure. Only for information. Can be any string.
     */
    private String author = "Unknown";
    /**
     * Date of figure creating. Only for information. Can be any string.
     */
    private String date = "Unknown";

    /**
     * Map of joints. For string indexing of them (by joints name).
     */
    private HashMap jointsMap = new HashMap();

    /**
     * Link to centralJoint.
     */
    public Joint centralJoint = null;

    /**
     * Constant to determine left parts of figure (hand and leg).
     */
    public static int LEFT = 0;
    /**
     * Constant to determine right parts of figure (hand and leg).
     */
    public static int RIGHT = 1;
    private Logger logger = null;
```

```
    /**
     * Constructor.
     */
    public Figure(String name, String authors, String date)
    {
        this.name = name;
        this.author = authors;
        this.date = date;
    }
```

```
/**
 * Returns figure name.
 */
public String getFigureName()
{
   return name;
}
```

```
/**
 * Returns figure author.
 */
public String getAuthor()
{
   return author;
}
```

```
/**
 * Returns figure date of creation.
 */
public String getDate()
{
   return date;
}
```

```
/**
 * Set joints vector
 */
public void setJoints(Vector joints)
{
   jointsMap.clear();

   for (int i = 0; i < joints.size(); i++)
   {
      Joint j = (Joint) joints.elementAt(i);
      jointsMap.put(j.getJointName(), j);
   }
}
```

```
/**
 * Returns number of joints
 */
public int getJointsSize()
{
   return jointsMap.size();
}
```

```
/**
 * Get joint by it's name
 */
public Joint getJointByName(String _name)
{
```

```
        return (Joint)jointsMap.get(_name);
    }
```

```
    public void setLogger(Logger logger)
    {
        this.logger = logger;
        for (Iterator i = jointsMap.keySet().iterator(); i.hasNext();)
        {
            Joint joint = (Joint)jointsMap.get(i.next());
            joint.setLogger(logger);
        }
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\FigureLoader.java

```
/**
 * Created by IntelliJ IDEA.
 * User: chizhik
 * Date: 08.11.2002
 * Time: 11:58:21
 */
package ru.ezhiki.automat.human.logic;

import org.xml.sax.*;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;

import javax.vecmath.Vector3d;
import java.io.FileReader;
import java.util.*;


public class FigureLoader extends DefaultHandler
{
    private HashMap joints = new HashMap();
    private Vector jointsVector = new Vector();

    private Joint centralJoint = null;

    private String currentJointName = null;
    private Joint currentJoint = null;

    private Figure figure = null;
```

```
    public void startElement(String namespaceURI,
                             String localName,
                             String qName,
                             Attributes attr) throws SAXException
    {
        if (localName.equals("figure"))
        {
            figure = new Figure(attr.getValue("name"), attr.getValue("authors"), attr.getValue("date"));

            joints.clear();
        }
```

```
    else if (localName.equals("joint"))
    {
       currentJointName = new String(attr.getValue("name"));
       currentJoint = new Joint();

       currentJoint.name = currentJointName;

       if (!attr.getValue("parent").equals("null"))
          currentJoint.parent = (Joint) joints.get(attr.getValue("parent"));

       currentJoint.m = new Double(attr.getValue("mass")).doubleValue();
       currentJoint.length = new Double(attr.getValue("length")).doubleValue();
    }
    else if (localName.equals("location"))
    {
       currentJoint.loc.x = new Double(attr.getValue("x")).doubleValue();
       currentJoint.loc.y = new Double(attr.getValue("y")).doubleValue();
       currentJoint.loc.z = new Double(attr.getValue("z")).doubleValue();
    }
    else if (localName.equals("massCentre"))
    {
       currentJoint.mLoc.x = new Double(attr.getValue("x")).doubleValue();
       currentJoint.mLoc.y = new Double(attr.getValue("y")).doubleValue();
       currentJoint.mLoc.z = new Double(attr.getValue("z")).doubleValue();
    }
    else if (localName.equals("angles"))
    {
       currentJoint.angles.x = new Double(attr.getValue("x")).doubleValue()/180*Math.PI;
       currentJoint.angles.y = new Double(attr.getValue("y")).doubleValue()/180*Math.PI;
       currentJoint.angles.z = new Double(attr.getValue("z")).doubleValue()/180*Math.PI;
    }
    else if (localName.equals("minAngles"))
    {
       currentJoint.minAngles.x = new Double(attr.getValue("x")).doubleValue()/180*Math.PI;
       currentJoint.minAngles.y = new Double(attr.getValue("y")).doubleValue()/180*Math.PI;
       currentJoint.minAngles.z = new Double(attr.getValue("z")).doubleValue()/180*Math.PI;
    }
    else if (localName.equals("maxAngles"))
    {
       currentJoint.maxAngles.x = new Double(attr.getValue("x")).doubleValue()/180*Math.PI;
       currentJoint.maxAngles.y = new Double(attr.getValue("y")).doubleValue()/180*Math.PI;
       currentJoint.maxAngles.z = new Double(attr.getValue("z")).doubleValue()/180*Math.PI;
    }
}
```

```
public void endElement(String namespaceURI,
                       String localName,
                       String qName) throws SAXException
{
    if (localName.equals("figure"))
    {
       System.gc();

       figure.setJoints(jointsVector);

       figure.centralJoint = centralJoint;
    }
    else if (localName.equals("joint"))
    {
       Joint added = null;
```

```
            if (currentJoint.parent != null)
            {
               added = currentJoint.parent.addChild(currentJoint);
               added.name = currentJointName;
               joints.put(currentJointName, added);
               jointsVector.add(added);
            }
            else
            {
               added = new Joint(null,
                     (Vector3d) currentJoint.loc.clone(),
                     currentJoint.length,
                     currentJoint.angles,
                     currentJoint.minAngles,
                     currentJoint.maxAngles,
                     currentJoint.m,
                     currentJoint.mLoc);
               added.name = currentJointName;
               joints.put(currentJointName, added);
               jointsVector.add(added);
            }

            if (currentJointName.equals("centralJoint"))
               centralJoint = added;
         }
      }
```

```
      public FigureLoader()
      {
      }
```

```
      public Figure loadFile(String fName)
      {
         try
         {
            XMLReader reader = XMLReaderFac-
tory.createXMLReader("org.apache.crimson.parser.XMLReaderImpl");
            reader.setContentHandler(this);
            reader.parse(new InputSource(new FileReader(fName)));
         }
         catch (SAXParseException e)
         {
            System.out.println("Error at line: " + e.getLineNumber() + ", column: " + e.getColumnNumber() + ",
Id: " + e.getPublicId());
            e.printStackTrace();
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }

         return figure;
      }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\Joint.java

```
/*
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: 20.10.2002
 * Time: 11:49:37
 */
package ru.ezhiki.automat.human.logic;

import ru.ezhiki.utils.MathUtils;
import ru.ezhiki.utils.Signaller;

import javax.vecmath.*;
import javax.media.j3d.*;
import java.util.Vector;
import java.util.Date;
import java.util.logging.Logger;

/**
 * The main class. It holds a joint representation. Joint is a stick, that is connected to other joints.
 * It turns, moves with the help of a simple automat. Joint is a part of Figure.
 * @see Figure
 * @see Actions
 */
public class Joint extends Thread
{
    /**
     * Name of the joint (To search it).
     */
    public String name = "";

    /**
     * Parent of this Joint. Central joint has no parent.
     */
    public Joint parent = null;

    /**
     * Children.
     */
    public Vector children = null;

    /**
     * length of the joint.
     */
    public double length = 0;

    /**
     * Minimum angles of the joint.
     */
    public Vector3d minAngles = new Vector3d(0, 0, 0);
    /**
     * Maximum angles of the joint.
     */
    public Vector3d maxAngles = new Vector3d(0, 0, 0);

    /**
     * Direction of the joint. X, Y - normal angles, Z - turn angle. These are relative, local angles.
     */
    public Vector3d angles = new Vector3d(0, 0, 0);

    /**
     * Position of the joint.
     */
```

```
    public Vector3d loc = new Vector3d(0, 0, 0);

    /**
     * Current settings of the angle.
     */
    private Vector3d curAngles = new Vector3d(0, 0, 0);
    /**
     * Current settings of the position.
     */
    private Vector3d curLoc = new Vector3d(0, 0, 0);

    /**
     * Mass centre of the joint.
     */
    public double m = 0;

    /**
     * Coordinates of the joint's mass centre
     */
    public Vector3d mLoc = null;

    /**
     * Java3D Group. Holds node and transform.
     */
    private BranchGroup branch = null;
    /**
     * Holds local transform of the joint.
     */
    private TransformGroup transform = null;

    /**
     * Delay between iterations while moving.
     */
    private static long delay = 4;

    /**
     * A variable for working (moving) signalling. Each dot is a working joint. If there are no dots - there are no
working joints
     */
    private Signaller signaller = Signaller.getSignaller();

    /**
     * Speed of turning. Can be changed during moving. In radians per second.
     */
    private double speedTurn = MathUtils.deg2rad(15);

    /**
     * Speed of translating. Can be changed during moving. In location units per second.
     */
    private double speedTranslate = 2;

    /**
     * Timer for corrent moving speed calculation.
     */
    private double time = -1;

    /**
     * Viscosity of connection to parent. More viscosity means harder joint rotation about parent. Must be >= 0, <= 1.
     * 1 means, that this joint is connected to the parent so, that it can't rotate relatively to it. Otherwise, 0 means, that
     * it can rotate without any force from the parent.
     */
    private double viscosity = 0;
```

```
  /**
   * Inertia of the joint's end (???). How it reacts to the parent's moving. 0 means, that it reacts momentically, 1 - it doesn't
   * react (I mean, that the connection doesn't brake, but the other end of this joint does'n move)
   */
  private double inertia = 0;

  /**
   * Automat A0 state
   */
  private int y0 = 0;

  private Vector3d dA = new Vector3d();
  private Vector3d dL = new Vector3d();
  private Vector3d tA = new Vector3d();
  private Vector3d tL = new Vector3d();
  private Logger logger = null;
```

```
  /**
   * Constructor
   */
  public Joint(
        Joint parent,
        Vector3d loc,
        double length,
        Vector3d angles,
        Vector3d minAngles,
        Vector3d maxAngles,
        double m,
        Vector3d mLoc)
  {
     this.parent = parent;
     this.length = length;
     this.minAngles = minAngles;
     this.maxAngles = maxAngles;
     this.angles = angles;
     this.curAngles = (Vector3d)this.angles.clone();
     this.loc = loc;
     this.curLoc = (Vector3d)this.loc.clone();
     this.m = m;
     this.mLoc = mLoc;

     children = new Vector();

     if (this.mLoc == null)
     {
        this.mLoc = new Vector3d(0, 0, length/2);
        this.mLoc.add(curLoc);
     }

     setPriority(Thread.MIN_PRIORITY);

     start();
  }
```

```
  /**
   * Default constructor
   */
  public Joint()
```

```
{
   this.parent = null;
   this.length = 5;
   this.m = 0;
   this.mLoc = null;
   children = new Vector();
}
```

```
/**
 * Main function of the thread
 */
public void run()
{
   while (true)
   {
      try
      {
         yield();
         sleep(delay);
      }
      catch (InterruptedException e)
      {
         e.printStackTrace();
      }

      if (x0_0() == true)
         A0();
   }
}
```

```
/**
 * Joint automat implementation.
 */
void A0()
{
   if (y0 == 0 && logger != null)
      logger.info(">>> A0 (Сустав \"" + name + "\")");

   switch (y0)
   {
      case 0:
         if (x0_0() == true) { z0_0();          y0 = 1; }
      break;
      case 1:
         z0_1();
         if (x0_1() == true) { z0_2();          y0 = 0; }
      break;
   }

   if (y0 == 0 && logger != null)
      logger.info("<<< A0 (Сустав \"" + name + "\")");
}
```

```
private boolean x0_0()
{
   return (curLoc.x != loc.x || curLoc.y != loc.y || curLoc.z != loc.z ||
               curAngles.x != angles.x || curAngles.y != angles.y || curAngles.z != angles.z);
```

```
}
```

```
private boolean x0_1()
{
   return finishedUpdating(dA, tA, dL, tL);
}
```

```
private void z0_0()
{
   if (logger != null)
      logger.info("    A0->z2_1 (Начало движения сустава \"" + name +  "\" ...)");
   time = new Date().getTime();
   z0_1();
}
```

```
private void z0_1()
{
   if (logger != null)
      logger.info(" %  A0->z2_1 (Перемещение сустава\"" + name +  "\" ...)");
   smallMove();
   update();
}
```

```
private void z0_2()
{
   if (logger != null)
      logger.info("    A0->z2_1 (Завершение движения сустава\"" + name +  "\")");
   signaller.removeSignal();
}
```

```
/**
 * Iterates angles during moving. And checks, did we reach the needed angles and location.
 */
private void smallMove()
{
   long curTime = new Date().getTime();
   double difference = (curTime - time)/1000.0;
   time = curTime;

   if (difference == 0)
      return;

   dA = (Vector3d)angles.clone();
   dA.sub(curAngles);
   tA = new Vector3d(speedTurn, speedTurn, speedTurn);

   dL = (Vector3d)loc.clone();
   dL.sub(curLoc);
   tL = new Vector3d(speedTranslate, speedTranslate, speedTranslate);

   if (dA.x != 0 || dA.y != 0 || dA.z != 0)
   {
      if (dA.x < 0)
```

```
            tA.x *= -1;
        if (dA.y < 0)
            tA.y *= -1;
        if (dA.z < 0)
            tA.z *= -1;

        tA.scale(difference);

        curAngles.x += updatedParameter(dA.x, tA.x);
        curAngles.y += updatedParameter(dA.y, tA.y);
        curAngles.z += updatedParameter(dA.z, tA.z);

        Vector3d checkedAngles = new Vector3d();

        checkedAngles.x = checkBounds(curAngles.x, maxAngles.x, minAngles.x);
        checkedAngles.y = checkBounds(curAngles.y, maxAngles.y, minAngles.y);
        checkedAngles.z = checkBounds(curAngles.z, maxAngles.z, minAngles.z);

        if (checkedAngles.x != curAngles.x)
        {
            dA.x = 0;
            curAngles.x = checkedAngles.x;
        }
        if (checkedAngles.y != curAngles.y)
        {
            dA.y = 0;
            curAngles.y = checkedAngles.y;
        }
        if (checkedAngles.z != curAngles.z)
        {
            dA.z = 0;
            curAngles.z = checkedAngles.z;
        }
    }

    if (dL.x != 0 || dL.y != 0 || dL.z != 0)
    {
        if (dL.x < 0)
            tL.x *= -1;
        if (dL.y < 0)
            tL.y *= -1;
        if (dL.z < 0)
            tL.z *= -1;

        tL.scale(difference);

        curLoc.x += updatedParameter(dL.x, tL.x);
        curLoc.y += updatedParameter(dL.y, tL.y);
        curLoc.z += updatedParameter(dL.z, tL.z);
    }
}
```

```
/**
 * Checks, whether first parameter is beetween second and third
 * @param x test parameter
 * @param maxX upper bound
 * @param minX lower bound
 * @return if minX <= x <= maxX, then it returns x, either - one of bounds
 */
private double checkBounds(double x, double maxX, double minX)
{
```

```
        if (x > maxX)
            x = maxX;
        else if (x < minX)
            x = minX;

        return x;
    }
```

```
    /**
     * Indicates, that angles and location do not need update more.
     */
    private boolean finishedUpdating(Vector3d dA, Vector3d tA, Vector3d dL, Vector3d tL)
    {
        if (Math.abs(dA.x) > Math.abs(tA.x) || Math.abs(dA.y) > Math.abs(tA.y) || Math.abs(dA.z) > Math.abs(tA.z)
||
                Math.abs(dL.x) > Math.abs(tL.x) || Math.abs(dL.y) > Math.abs(tL.y) || Math.abs(dL.z) >
Math.abs(tL.z))
            return false;
        else
        {
            curLoc.set(loc);
            curAngles.set(angles);
            return true;
        }
    }
```

```
    /**
     * Choose, which absolute value is less. And returns it.
     */
    private double updatedParameter(double dX, double tX)
    {
        if (Math.abs(dX) > Math.abs(tX))
            return tX;
        else
            return dX;
    }
```

```
    /**
     * Updates transform (for visualizing), children and parent.
     */
    private void update()
    {
        transform.setTransform(getCurrentTransform());
        if (parent != null)
            parent.update();
    }
```

```
    /**
     * Returns Node for attaching into Java3D Scene.
     */
    public Node get3dShape()
    {
        branch = new BranchGroup();

        transform = new TransformGroup(getCurrentTransform());
```

```
        transform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

        branch.addChild(transform);

        Material material = new Material();
        material.setLightingEnable(true);
        material.setDiffuseColor(0.8f, 0.8f, 0.8f);

        Appearance appearanceLines = new Appearance();
        appearanceLines.setColoringAttributes(new ColoringAttributes(new Color3f(0.5f, 0.5f, 0.5f), ColoringAt-
tributes.SHADE_GOURAUD));
        appearanceLines.setLineAttributes(new LineAttributes(2, LineAttributes.PATTERN_SOLID, true));
        appearanceLines.setMaterial(material);

        Appearance appearancePoints = new Appearance();
        appearancePoints.setColoringAttributes(new ColoringAttributes(new Color3f(0.5f, 0.5f, 0.5f), ColoringAt-
tributes.SHADE_GOURAUD));
        appearancePoints.setPointAttributes(new PointAttributes((int)(m/2 > 4 ? m/2 : 4), true));
        appearancePoints.setMaterial(material);

        IndexedPointArray points = new IndexedPointArray(1, GeometryArray.COORDINATES | GeometryAr-
ray.COLOR_3, 1);

        points.setColorIndices(0, new int[] {0});
        points.setColors(0, new Color3f[] {new Color3f(0.1f, 0.1f, 1)});
        points.setCoordinateIndices(0, new int[] {0});
        points.setCoordinates(0, new Point3d[] {new Point3d(mLoc)});

        IndexedLineArray lines =
              new IndexedLineArray(2, GeometryArray.COORDINATES | GeometryArray.COLOR_3, 2);

        Vector3d drawLoc = (Vector3d)curLoc.clone();
        Vector3d drawEndLoc = (Vector3d)curLoc.clone();
        drawEndLoc.add(new Vector3d(0, 0, length));

        int pntsIndex[] = {0, 1};
        Point3d pnts[] = {new Point3d(drawLoc), new Point3d(drawEndLoc)};
        int clrsIndex[] = {0, 1};
        Color3f clrs[] = {new Color3f(1, 0.1f, 0.1f), new Color3f(0.1f, 1, 0.1f)};

        lines.setColorIndices(0, clrsIndex);
        lines.setColors(0, clrs);
        lines.setCoordinateIndices(0, pntsIndex);
        lines.setCoordinates(0, pnts);

        transform.addChild(new Shape3D(lines, appearanceLines));
        transform.addChild(new Shape3D(points, appearancePoints));

        for (int i = 0; i < children.size(); i++)
        {
           Joint joint = (Joint) children.elementAt(i);
           transform.addChild(joint.get3dShape());
        }

        return branch;
    }
```

```
    /**
     * Current transform of this joint.
     * @return transform matrix
     */
```

```
  private Transform3D getCurrentTransform()
  {
     Transform3D toView = new Transform3D();
     Quat4d q = quatFromEuler();

     Vector3d coordStart = null;
     if (parent != null)
     {
        coordStart = (Vector3d)parent.curLoc.clone();
        coordStart.add(new Vector3d(0, 0, parent.length));
        coordStart.add(curLoc);
     }
     else
     {
        coordStart = (Vector3d)curLoc.clone();
     }

     toView.setRotation(q);
     toView.setTranslation(coordStart);

     return toView;
  }
```

```
  /**
   * Returns quaternion, that is constructed from angles field.
   */
  private Quat4d quatFromEuler()
  {
     double roll = curAngles.x;
     double pitch = curAngles.y;
     double yaw = curAngles.z;

     Quat4d qRoll = new Quat4d(Math.sin(roll/2), 0, 0, Math.cos(roll/2));
     Quat4d qPitch = new Quat4d(0, Math.sin(pitch/2), 0, Math.cos(pitch/2));
     Quat4d qYaw = new Quat4d(0, 0, Math.sin(yaw/2), Math.cos(yaw/2));

     Quat4d q = new Quat4d();

     q = qYaw;
     q.mul(qPitch);
     q.mul(qRoll);

     return q;
  }
```

```
  /**
   * Adds a child to this joint.
   * @return a joint, that was added. Notice, that it is not the same joint, that was passed as a parameter.
   */
  public Joint addChild(Joint aJoint)
  {
     Joint automat = new Joint(this, aJoint.loc, aJoint.length, aJoint.angles, aJoint.minAngles,
aJoint.maxAngles, aJoint.m, aJoint.mLoc);
     children.add(automat);

     return automat;
  }
```

```
/**
 * Set x-angle
 */
public void setAngleX(double _x)
{
   if (angles.x != _x)
      angles.x = _x;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set y-angle
 */
public void setAngleY(double _y)
{
   if (angles.y != _y)
      angles.y = _y;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set z-angle
 */
public void setAngleZ(double _z)
{
   if (angles.z != _z)
      angles.z = _z;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set x coordinate of the location
 */
public void setLocX(double _x)
{
   if (loc.x != _x)
      loc.x = _x;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set y coordinate of the location
 */
public void setLocY(double _y)
{
   if (loc.y != _y)
      loc.y = _y;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set z coordinate of the location
 */
public void setLocZ(double _z)
{
   if (loc.z != _z)
      loc.z = _z;
   else
      signaller.removeSignal();
}
```

```
/**
 * Set all angles and location.
 */
public void setAnglesAndLocation(Vector3d a, Vector3d l)
{
   boolean wasSet = false;

   if (a != null && (angles.x != a.x || angles.y != a.y || angles.z != a.z))
   {
      angles = (Vector3d)a.clone();
      wasSet = true;
   }
   if (l != null && (loc.x != l.x || loc.y != l.y || loc.z != l.z))
   {
      loc = (Vector3d)l.clone();
      wasSet = true;
   }

   if (!wasSet)
      signaller.removeSignal();
}


/**
 * Set speeds of turning and translation.
 */
public void setSpeeds(String speedTurn, String speedTranslate)
{
   setSpeedTurn(speedTurn);
   setSpeedTranslate(speedTranslate);
}


/**
 * Set speed of translation.
 */
public void setSpeedTranslate(String speedTranslate)
{
   if (speedTranslate != null)
   {
      try
      {
         this.speedTranslate = new Double(speedTranslate).doubleValue();
      }
      catch (NumberFormatException e)
```

```
        {
            this.speedTranslate = 2;
        }
    }
    else
    {
        this.speedTranslate = 2;
    }
}
```

```
/**
 * Set speed of turning.
 */
public void setSpeedTurn(String speedTurn)
{
    if (speedTurn != null)
    {
        try
        {
            this.speedTurn = MathUtils.deg2rad(new Double(speedTurn).doubleValue());
        }
        catch (NumberFormatException e)
        {
            this.speedTurn = MathUtils.deg2rad(15);
        }
    }
    else
    {
        this.speedTurn = MathUtils.deg2rad(15);
    }
}
```

```
public String getJointName()
{
    return name;
}
```

```
public void setLogger(Logger logger)
{
    this.logger = logger;
}
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\JointAction.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 9:57:09 PM
 */
package ru.ezhiki.automat.human.logic;

import javax.vecmath.Vector3d;

public class JointAction
{
   private String name = "";
   private String speedTurn = "15";
   private String speedTranslate = "5";
   private Vector3d loc = null;
   private Vector3d angles = null;
```

```
   public JointAction(String name, String speedTurn, String speedTranslate)
   {
      this.name = name;
      this.speedTurn = speedTurn;
      this.speedTranslate = speedTranslate;
      loc = null;
      angles = null;
   }
```

```
   public JointAction(String name)
   {
      this.name = name;
      loc = null;
      angles = null;
   }
```

```
   public String getName()
   {
      return name;
   }
```

```
   public String getSpeedTurn()
   {
      return speedTurn;
   }
```

```
   public String getSpeedTranslate()
   {
      return speedTranslate;
```

```
    }
```

```
    public Vector3d getLocation()
    {
        return loc;
    }
```

```
    public Vector3d getAngles()
    {
        return angles;
    }
```

```
    public void setLocation(double x, double y, double z)
    {
        if (loc == null)
            loc = new Vector3d(0, 0, 0);

        loc.x = x;
        loc.y = y;
        loc.z = z;
    }
```

```
    public void setAngles(double x, double y, double z)
    {
        if (angles == null)
            angles = new Vector3d(0, 0, 0);

        angles.x = x;
        angles.y = y;
        angles.z = z;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\logic\Position.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 9:55:31 PM
 */
package ru.ezhiki.automat.human.logic;

import java.util.HashMap;
import java.util.Set;

public class Position
{
    private String name = "";
    private HashMap jointActions = null;
```

```
    public Position(String name)
    {
        this.name = name;
        jointActions = new HashMap();
    }
```

```
    public String getName()
    {
        return name;
    }
```

```
    public void addJointAction(JointAction jointAction)
    {
        jointActions.put(jointAction.getName(), jointAction);
    }
```

```
    public Set getJointNames()
    {
        return jointActions.keySet();
    }
```

```
    public JointAction getJointAction(String jName)
    {
        return (JointAction)jointActions.get(jName);
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\vis\AutomatCanvas.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: 04.11.2002
 * Time: 12:28:39
 */
package ru.ezhiki.automat.human.vis;

import ru.ezhiki.automat.human.logic.Figure;

import javax.media.j3d.*;
import javax.vecmath.Vector3d;

import java.awt.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.MouseMotionListener;

import com.sun.j3d.utils.universe.SimpleUniverse;

public class AutomatCanvas
```

```
{
   private Figure figure = null;

   private SimpleUniverse universe = null;
   private BranchGroup rootBranch = null;
   private TransformGroup rootTransform = null;

   public Canvas3D canvas3d = null;

   private double scale = 0.04;
   private double rotateX = Math.PI/2;
   private double rotateY = 0;
   private double rotateZ = 0;
   private double moveX = 0;
   private double moveY = 1;
   private double moveZ = 0;

   private int mousePosX = -1;
   private int mousePosY = -1;
```

```
   public AutomatCanvas(Figure figure)
   {
      canvas3d = new Canvas3D(GraphicsEnvironment.getLocalGraphicsEnvironment().
            getDefaultScreenDevice().getBestConfiguration(new GraphicsConfigTemplate3D()));
      this.figure = figure;

      canvas3d.addMouseListener(new MouseListener()
      {
         public void mouseClicked(MouseEvent e) {}

         public void mousePressed(MouseEvent e)
         {
            mousePosX = e.getX();
            mousePosY = e.getY();
         }

         public void mouseReleased(MouseEvent e) {}

         public void mouseEntered(MouseEvent e) {}

         public void mouseExited(MouseEvent e) {}
      });

      canvas3d.addMouseMotionListener(new MouseMotionListener()
      {
         public void mouseDragged(MouseEvent e)
         {
            if (mousePosX == -1 || mousePosY == -1)
            {
               mousePosX = e.getX();
               mousePosY = e.getY();
               return;
            }

            rotateZ += (mousePosX - e.getX())*0.01;

            rootTransform.setTransform(getCurrentTransform());

            mousePosX = e.getX();
            mousePosY = e.getY();
         }
```

```
        public void mouseMoved(MouseEvent e) {}
    });
  }
```

```
  public void redraw()
  {
  }
```

```
  public void updateCanvas()
  {
    universe = new SimpleUniverse(canvas3d);
    universe.getViewingPlatform().setNominalViewingTransform();

    canvas3d.stopRenderer();
    universe.addBranchGraph(createSurfaceGroup());
    canvas3d.startRenderer();
  }
```

```
  private BranchGroup createSurfaceGroup()
  {
    rootBranch = new BranchGroup();

    rootTransform = new TransformGroup(getCurrentTransform());
    rootTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    if (figure != null)
      rootTransform.addChild(figure.centralJoint.get3dShape());

    rootBranch.addChild(rootTransform);
    rootBranch.compile();

    return rootBranch;
  }
```

```
  private Transform3D getCurrentTransform()
  {
    Transform3D rotX = new Transform3D();
    Transform3D rotY = new Transform3D();
    Transform3D rotZ = new Transform3D();

    rotX.rotX(rotateX);
    rotY.rotY(rotateY);
    rotZ.rotZ(rotateZ);

    Transform3D toView = new Transform3D();
    toView.mul(rotX);
    toView.mul(rotZ);
//      toView.mul(rotY);

    toView.setScale(scale);
    toView.setTranslation(new Vector3d(moveX, moveY, moveZ));
    return toView;
  }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\automat\human\vis\AutomatFrame.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: 04.11.2002
 * Time: 12:28:28
 */
package ru.ezhiki.automat.human.vis;

import ru.ezhiki.automat.human.logic.Figure;
import ru.ezhiki.automat.human.logic.Actions;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.logging.Logger;

public class AutomatFrame extends JFrame
{
   private AutomatCanvas canvas = null;
   private Figure figure = null;
   private Actions actions = null;
   private Logger logger = null;

   private JButton buttonStart = new JButton(" Start ");
```

```
   public AutomatFrame(Figure figure, Actions actions, Logger logger) throws HeadlessException
   {
      setTitle("Joint Automat");

      this.logger = logger;
      this.figure = figure;
      this.actions = actions;
      if (actions != null)
         actions.setFigure(figure);

      figure.setLogger(logger);
      actions.setLogger(logger);

      init3d();
      initLayout();
      initActions();

      canvas.updateCanvas();

      pack();
      setSize(300, 500);

      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = getSize();
      if (frameSize.height > screenSize.height)
      {
         frameSize.height = screenSize.height;
      }
      if (frameSize.width > screenSize.width)
      {
```

```
      frameSize.width = screenSize.width;
    }
    setLocation((screenSize.width - frameSize.width)/2, (screenSize.height - frameSize.height)/2);
}
```

```
private void initLayout()
{
    getContentPane().setBackground(Color.black);
    getContentPane().add(buttonStart, BorderLayout.SOUTH);
    getContentPane().add(canvas.canvas3d, BorderLayout.CENTER);
}
```

```
private void initActions()
{
    canvas.canvas3d.addKeyListener(new KeyListener()
    {
        public void keyTyped(KeyEvent e)
        {
            if (e.getKeyChar() == '1')
            {
                actions.processActions();
            }
        }

        public void keyPressed(KeyEvent e)
        {
        }

        public void keyReleased(KeyEvent e)
        {
        }
    });

    buttonStart.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            actions.processActions();
        }
    });
}
```

```
private void init3d()
{
    canvas = new AutomatCanvas(figure);
    canvas.canvas3d.setBackground(Color.black);
}
```

```
protected void processWindowEvent(WindowEvent e)
{
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
        exitWindow();
    }
```

```
    }
```

```
    private void exitWindow()
    {
        System.exit(0);
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\utils\AutomatSimpleFormatter.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: 21.01.2003
 * Time: 23:11:53
 * To change this template use Options | File Templates.
 */
package ru.ezhiki.utils;

import java.util.logging.Formatter;
import java.util.logging.LogRecord;
import java.sql.Date;

public class AutomatSimpleFormatter extends Formatter
{
    public AutomatSimpleFormatter()
    {
    }
```

```
    public String format(LogRecord record)
    {
        return record.getMessage() + "\n";
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\utils\Debug.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 13, 2002
 * Time: 7:51:11 PM
 */
package ru.ezhiki.utils;

public class Debug
{
    private static boolean debug = true;
    private static Debug debugVar = new Debug();
```

```
    private Debug()
```

```
    {
    }
```

```
    public static boolean isDebug()
    {
        return debug;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\utils\MathUtils.java

```
/**
 * Created by IntelliJ IDEA.
 * User: babaev
 * Date: Nov 12, 2002
 * Time: 4:10:15 PM
 */
package ru.ezhiki.utils;

public class MathUtils
{
    public static double deg2rad(double deg)
    {
        return deg/180*Math.PI;
    }
```

```
    public static double rad2deg(double rad)
    {
        return rad/Math.PI*180;
    }
}
```

X:\Programming\Java\Shalyto\src\ru\ezhiki\utils\Signaller.java

```
/**
 * Created by IntelliJ IDEA.
 * User: Alexander
 * Date: Nov 12, 2002
 * Time: 10:44:52 PM
 */
package ru.ezhiki.utils;

import java.util.logging.Logger;

/**
 * Class for synchronizing actions. It uses simple counter for storing number
 * of actions, now being executed. If number == 0, then no actions are currently
 * performed.
 *
 * It is made for correct actionList going. Because every action has some subactions
 * and these subactions can finish at different times.
 *
 * But i need next action to begin only after all previous action's subactions are finished.
```

```
 * That's where from this class started.
 *
 */
public class Signaller
{
   /**
    * Singleton instance.
    */
   private static Signaller signaller = new Signaller();

   /**
    * number of currently executing subactions
    */
   private Integer signalsCnt = new Integer(0);

   /**
    * Automat state.
    */
   private int y0 = 0;
   private Logger logger = null;
```

```
   /**
    * I need only one instance of signaller per
    * application, so constructor is private.
    */
   private Signaller()
   {
      logger = Logger.getLogger("ru.ezhiki.automat.human.logger");
   }
```

```
   /**
    * Getter for the singleton.
    * @return singleton instance.
    */
   public static Signaller getSignaller()
   {
      return signaller;
   }
```

```
   /**
    * Does the signaller has signals.
    * @return true, if number of signals more, than zeor, false otherwise.
    */
   public boolean hasSignals()
   {
      synchronized (signalsCnt)
      {
         if (signalsCnt.intValue() > 0)
            return true;
         else
            return false;
      }
   }
```

```
   /**
```

```
   * Put a signal.
   */
  public void addSignal()
  {
     A1(0);
  }
```

```
  /**
   * Remove a signal.
   */
  public void removeSignal()
  {
     A1(1);
  }
```

```
  /**
   * Clear signaller.
   */
  public synchronized void clear()
  {
     synchronized (signalsCnt)
     {
        signalsCnt = new Integer(0);
     }
  }
```

```
  /**
   * Standard overiding.
   * @return string, representing current number of sygnals.
   */
  public String toString()
  {
     synchronized (signalsCnt)
     {
        return signalsCnt.toString();
     }
  }
```

```
  /**
   * Signaller automat implementation.
   * @param e - event number
   */
  private void A1(int e)
  {
//      if (logger != null)
//         logger.info(">>> A1 (Signaller)");

     switch(y0)
     {
        case 0:
           if (e == 0) { z1_0();          y0 = 1; }
           else
           if (e == 1) {                      }
        break;
        case 1:
```

```
            if (e == 0) { z1_0();                  }
            else
            if (e == 1 && x1()) { z1_1();          }
            else
            if (e == 1 && !x1()) { z1_1();   y0 = 0; }
          break;
      }

//      if (logger != null)
//        logger.info("<<< A1 (Signaller)");
   }
```

```
   /**
    * Is there only one signal?
    * @return
    */
   private boolean x1()
   {
      return (signalsCnt.intValue() > 1);
   }
```

```
   /**
    * Add a signal
    */
   private void z1_0()
   {
      if (logger != null)
         logger.info(" +  A1->z1_0 (Добавлен сигнал)");

      synchronized (signalsCnt)
      {
         signalsCnt = new Integer(signalsCnt.intValue() + 1);
      }
   }
```

```
   /**
    * Remove a signal
    */
   private void z1_1()
   {
      if (logger != null)
         logger.info(" -  A1->z1_0 (Удален сигнал)");

      synchronized (signalsCnt)
      {
         if (signalsCnt.intValue() == 0)
            return;
         else if (signalsCnt.intValue() == 1)
            signalsCnt = new Integer(0);
         else
            signalsCnt = new Integer(signalsCnt.intValue() - 1);
      }
   }
```

```
   public void setLogger(Logger logger)
```

```
    {
        this.logger = logger;
    }
}
```