ИСПОЛЬЗОВАНИЕ JAVABEANS-КОМПОНЕНТ В 3D-МОДЕЛИРОВАНИИ

Е.М. Гринкруг,
доцент кафедры «Управление разработкой программного обеспечения» факультета бизнес-информатики Государственного университета — Высшей школы экономики, кандидат технических наук,
e-mail: egrinkrug@hse.ru.
Адрес: 105187, г. Москва, ул. Кирпичная, д. 33/5.

В статье рассматривается использование компонентной модели JavaBeans как основы реализации программных средств 3D-моделирования и визуализации. Моделирование 3D-объектов основано на реализации JavaBeans-компонент, функционально соответствующих подмножеству базовых типов объектов, используемых в стандартных языках 3D-моделирования, в сочетании с дополнительными компонентами, расширяющими возможности таких языков. Практическим результатом работы является библиотека JavaBeans-компонент (lightweight, 100% Java), которая используется в Java-апплетах и приложениях.

Ключевые слова: JavaBeans, VRML, 3D апплет.

1. Введение

Целью данной работы является исследование возможностей компонентной модели Java Beans для организации 3D-моделирования, а именно – для поддержки поведения объектов 3D-модели, собранной из JavaBeans-компонент, и их визуализации.

Язык Java первенствует по популярности среди языков программирования [1]. Значительное количество исследовательских проектов реализуется с применением Java, в том числе – программных проектов, связанных с 3D-моделированием. Java располагает библиотекой Java3D [2], которая поддерживает поведение и визуализацию 3D-объектов, однако эта библиотека имеет ряд недостатков: имеет значительный объем, недостаточно документирована, и не вполне соответствует принципу мультиплатформенности самой Java-машины.

В данной работе рассматривается реализация 3D-моделирования с помощью стандартных JavaBeans-компонент, и обсуждаются возможности компонентного расширения средств 3D-моделирования. В качестве примера базовых средств описания 3D-модели использовано подмножество типов объектов, функционально соответствующих основным типам узлов языка VRML [3] – декларативного языка моделирования виртуальной реальности, являющегося стандартом ISO. Это подмножество дополнено компонентами, не имеющими аналогов в VRML и расширяющими его возможности. Практическим результатом работы является библиотека JavaBeans-компонент, с помощью которой в любом контейнере, функционально эквивалентном BDK [4, 5], легко осуществляется создание, функционирование и визуализация 3D-моделей, а также обеспечивается

использование этих моделей в Java-апплетах и Java-приложениях.

В работе не ставилась цель полной, соответствующей стандарту, реализации языка 3D-моделирования (такого, как язык VRML или его преемник X3D [6]). Реализованное подмножество базовых средств, дополненное отсутствующими в этих языках компонентами, достаточно для апробации рассматриваемого подхода к 3D-моделированию с помощью JavaBeans-компонент, оценки достоинств и недостатков этого подхода, а также для практического применения разработанной библиотеки компонент.

2. Моделирование с помощью Java Beans-компонент

Согласно [4], «JavaBean – это повторно используемый программный компонент, которым можно манипулировать визуально в инструменте сборки». Изначально, с появлением спецификации JavaBeans, встречалось дополнение о том, что компонентная модель JavaBeans – это единственная компонентная модель для Java-машин; потом появились и другие, предъявляющие к компонентам более специфические требования .

Компонент JavaBean – это Java-класс, который может быть инстанциирован вызовом конструктора без параметров для использования в любом контейнере, и который является сериализуемым. Все остальные требования к разработке компонент JavaBeans являются соглашениями (design patterns), следование которым упрощает манипулирование экземплярами компонента внутри контейнера по сравнению с объектами обычных классов.

Требование сериализуемости стандартными средствами Java связано со способом сохранения результатов ассемблирования экземпляров компонент, предусмотренным при разработке спецификации JavaBeans. В настоящее время это требование не является актуальным, поскольку с развитием Java-платформы появились более удобные средства сохранения и восстановления состояния совокупности экземпляров компонент, основанные на использовании декларативных языков.

Основой компонентной модели JavaBeans является динамическая рефлекция и инроспекция классов, что позволяет в динамике получить информацию о методах (MethodDescriptors), свойствах компонент (PropertyDescriptors) и генерируемых событиях (EventSetDescriptors).

Одновременно со спецификацией компонентной модели JavaBeans был разработан Bean Development Kit (BDK) [4, 5] – инструмент, предназначенный для апробации возможностей компонентной модели JavaBeans вообще и конкретных JavaBeans-компонент, в частности. Эти возможности (в разной степени) поддерживаются различными средами разработки программного обеспечения на языке Java (IDEs). Одним их примеров IDE, где поддержка компонентной модели JavaBeans была реализована наиболее широко, являлся продукт IBM VisualAge for Java, впоследствии уступивший место IDE Eclipse. Основное, но не единственное, использование таких инструментов было связано с разработкой графических интерфейсов (GUI), так как практически все элементы Java GUI являются JavaBeans-компонентами.

Компонентная модель JavaBeans была разработана уже для первых версий JDK, широко применяется, начиная с JDK1.1, и получила развитие в последующих версиях JDK, где добавлялись новые возможности Java-платформ. Однако, BDK не претерпел существенных изменений с версии 1.1 (BeanBuilder [10], аналог BDK, привнес сохранение состояния сконфигурированных компонент в XML-файле и имеет GUI, основанный на пакете Swing.). Существуют многочисленные специализированные инструменты для работы с компонентами JavaBeans.

Исторически, различные инструментальные средства предоставляли разные способы использования компонентной модели JavaBeans, отличающиеся способом представления и использования результирующей композиции из компонент. Так, IBM VisualAge for Java использовал композицию экземпляров компонент, по мере ее визуального построения, для генерации исходного кода соответствующего композиции класса. Полученный код компилировался, и создавался новый («статически скомпилированный») составной компонент. При таком подходе, языком сборки компонент является язык их реализации (фактически имеет место визуальная автоматизация ручного кодирования с последующей компиляцией). Результирующая композиция компонент после компиляции может являться компонентом, экземпляры которого могут использоваться при построении новых композиций, и т.д.

Позднее, для связывания компонент стали использоваться декларативные языки, основанные на XML [7]. Применительно к GUI – это XUL [8], и

другие (в MS .NET в аналогичных целях используется XAML [9]). Результатом визуальной сборки компонент в инструментальном контейнере при этом может служить сериализованное в XML состояние экземпляра контейнера с собранными в нем экземплярами компонент, повторно используемое путем последующий десериализации. Подобные декларативные средства не требуют компиляции (достаточно простого парсера декларативного языка), однако они не приводят к созданию новых компонент как новых инстанциируемых типов объектов. В этом случае инстанциирование (составных) компонент для повторного использования заменяется клонированием (или клонированием через сериализацию/десериализацию) собранного экземпляра-образца (иногда называемого прототипом).

Имеется ряд инструментов моделирования, основанных на сходных принципах, но не использующих компонентную модель JavaBeans, с том числе DEVS [11] (и Java-реализация [12]), проект Ptolemy (с Java-реализацией Ptolemy II [13]). На сходных принципах основан популярный инструмент моделирования сетей Omnet++ [14]. Компонентная модель Fractal [15, 16] имеет средства определения составных компонент, в том числе для языка Java, но не опирается на общепринятую компонентную модель JavaBeans. С другой стороны, имеющиеся публикации об использовании компонентной модели JavaBeans в целях моделирования (например, [17, 18]) обходят стороной вопросы моделирования и визуализации 3D-объектов.

3. Моделирование 3D-объектов

VRML (Virtual Reality Modeling Language) – язык моделирования виртуальной реальности, ставший стандартом ISO/IEC 14772-1:1997, как и X3D (преемник VRML с использованием XML-синтаксиса (ISO/IEC19775-1), предполагают моделирование виртуальной реальности на основе взаимодействующих программных объектов, описывающих модель, ее поведение и 3D-представление.

Как отмечал один из создателей VRML [19] на заре его появления, совпавшего по времени с началом популярности языка Java, – «Java и VRML отлично дополняют друг друга. Java говорит о том, как объекты себя ведут, но почти ничего не говорит о том, как они выглядят. VRML в основном предназначен для определения внешнего вида объектов вне связи с их поведением. Можно сказать, что VRML отвечает за существование, а Java – за выполнение. Они нужны друг другу...».

В литературе и практике Java-программирования использование самой популярной компонентной модели JavaBeans для 3D-моделирования представлено недостаточно. Библиотека Java3D не уделяет должного внимания вопросам компонентности, как и реализация X3D на ее основе (Xj3D [2, 20]). Стандартные реализации VRML [21] могут поддерживать взаимодействие с Java-программами, предоставляя два вида интерфейсов: External Authoring Interface (EAI) и Script Authoring Interface (SAI). Интерфейс EAI обеспечивает управление 3D-моделями, инкапсулированными в понятии Browser. Интерфейс SAI позволяет встраивать в модель скрипты, реализованные как наследованные из базового класса Script Java-классы, способные управлять поведением моделей. Ограниченность связи 3D-моделей с Java-программами с помощью этих интерфейсов в популярных VRML-браузерах объясняется их реализацией вне Java-машины. Имеющиеся Java-реализации различных подмножеств VRML либо решают чисто практическую задачу визуализации 3D-моделей в Java-апплетах (BS Contact J 3D applet [22] и др.), либо используют нестандартные (proprietary) компонентные модели (как, например, Demicron WireFusion [23]). В последнее время интенсивно развивается JavaFX [24]. JavaFX Script [25] – декларативный язык, имеющий сходные с Miscrosoft Silverlight [26] цели и средства, идеологически близок с VRML, но компилируется в исполняемый Java байткод. Собственных средств 3D-моделирования и отображения JavaFX не имеет, но есть попытки их реализации с помощью Java3D [27].

После спада популярности VRML, пик которой наблюдался около 10 лет назад, появились многочисленные альтернативные форматы описания 3D-моделей [28, 29, 30]. Вместе с тем, VRML является полезной отправной точкой для исследования применимости компонентного подхода к 3D-моделированию и визуализации.

В VRML 3D-модель строится как совокупность объектов-узлов, образующих граф сцены (scene graph). Узлы являются экземплярами предопределенных типов (в VRML их более 50-ти), и типов, определяемых разработчиком модели. Имеется два способа определения новых типов узлов: 1) с помощью узлов-скриптов, определяющих реализацию на языках программирования (стандарт предусматривает JavaScript и/или Java), и 2) с помощью определения прототипов составных узлов.

Каждый объект-узел содержит набор типизированных полей (номенклатура типов полей зафик-

сирована стандартом). Все типы значений полей делятся на скалярные (хранят одно значение) и векторные (хранят векторы однотипных значений). Среди типов полей, наряду с типами-значениями, есть типы, позволяющие узлам ссылаться друг на друга, благодаря чему образуется граф сцены. Этот граф должен быть траверсируемым, то есть представлять собой направленный ациклический граф узлов (Directed Acyclic Graph, DAG). Граф может содержать узлы (или подграфы), на которые ссылаются более одного узла. Это позволяет использовать узлы и подграфы повторно в целях разделения общих данных. Имеющиеся типы узлов классифицируются по назначениям. Некоторые узлы непосредственно определяют результат 3D-визуализации, некоторые – косвенно, и некоторые могут не использоваться для визуализации.

Состав полей узла определяются его типом. Поле может содержать значение своего типа, и допускать только запись (входное поле узла, eventIn), только чтение и выработку события при изменении значения (выходное поле, eventOut), или поддерживать обе эти возможности одновременно. Помимо графа узлов сцены может существовать произвольный граф связей узлов по событиям (event routes). Сигнал изменения выходного поля одного узла может передаваться (объектами event route) на входное поле другого(других), если входное поле имеет тот же тип значений.

4. Реализация 3D-модели средствами JavaBeans-компонент

В рассматриваемой реализации каждый тип узла реализуется JavaBean-компонентом. Именованные поля узлов при этом соответствуют одноименным свойствам JavaBean'а, реализованным setter'ами и getter'ами, и описываемым стандартными PropertyDescriptor'ами (для векторных полей – IndexedPropertyDescriptor'ами). Реализация передачи событий обеспечивается за счет того, что выходные поля узлов реализуются как связываемые (bindable) свойства bean-компонент, передающие PropertyChangeEvent(s).

При этом мы несколько отходим от принятых в VRML правил, но получаем полезные преимущества. В частности, обеспечивается универсальная реализация парсера VRML-файлов, основанная на возможностях JavaBeans. Кроме того, мы никак не ограничиваем возможности расширения номенклатуры типов узлов-компонент и типов значений их свойств. BeanBox – контейнер при этом может играть роль визуального инструмента моделирования и отладки. Компонентность позволяет оптимально распространять реализацию конкретной модели, предоставляя не всю номенклатуру типов узлов, а только используемые в ней классы. Недостатком данного подхода к реализации 3D-модели, вытекающим из «полустатической» природы стандартной компонентной модели JavaBeans, являются ограничения возможностей реализации декларативно определяемых составных типов (эквивалентов прототипам VRML).

Граф модели, состоящий из экземпляров компонент, может строиться программно, визуально (в BeanBox-подобном контейнере) или считываться из своего сериализованного представления. Поскольку речь идет о воссоздании графа объектов – экземпляров JavaBeans-компонент, можно использовать все способы сериализации и десериализации, имеющиеся в Java (встроенные, средства XML-сериализации/десериализации и т.п.). Мы остановимся на считывании модели из стандартного VRML-формата, что позволяет использовать арсенал VRML-ориентированных инструментов и готовых 3D-объектов в этом формате.

5. Реализация парсера для подмножества VRML

Возможны разные способы десериализации графа сцены из декларативного описания. Одним из простейших подходов является наличие метода десериализации в каждом классе реализации компонент. Однако, этот способ требует от компонент реализации предопределенного интерфейса (что уже является ограничением), и соответствующего увеличения кода каждого компонента.

Начиная с JDK 1.4, объекты, составленные из экземпляров JavaBeans-компонент, могут подвергаться XML-сериализации/десериализации без реализации соответствующих методов в самих компонентах [31]. Такой механизм предполагает, что классы реализации компонент предоставляют методы чтения (для сериализации) и записи (для десериализации) значений всех свойств компонент. Средства JDK (XMLEncoder/XMLDecoder) предполагают, что десериализуемый XML-файл содержит последовательность действий, выполнение которых дает представление составного объекта в памяти. XML-текст при этом довольно многословен. VRML-текст лаконичнее, так как декларативное описание в VRML-файле определяет не последовательность действий для получения графа объектов, а собственно состояние графа.

Для считывания из VRML-файла графа сцены, состоящего их экземпляров стандартных JavaBeans-компонент, реализующих узлы VRML, мы используем другой подход к реализации парсера. Парсер управляется самим исходным текстом и структурой JavaBeans-компонент. Исходный текст считывается до определения имени типа узла, и инстанциируется соответствующий компонент. Далее выполняется цикл считывания имеющихся в исходном файле пар <имя, значение>, ограниченных рамками декларации узла. Каждое имя понимается как имя свойства инстанциированного компонента (propertyName), которое имеет известный для компонента тип (propertyType) и которому надо присвоить (вызовом setter'а) десериализованное значение. Дальнейшее считывание значения управляется типом свойства. Значения типов, являющихся массивами, считываются циклом считывания элементов массива. Значения известных скалярных типов свойств считываются:

✦ общими методами получения значений примитивных типов языка Java из строки;

✦ получением из входного текста значений параметров для вызова конструктора, который даст требуемое значение свойства (каждый такой параметр считывается рекурсивным применением процедуры чтения значения известного типа из входного текста);

✦ рекурсивным применением всей описанной выше процедуры, если типом значения свойства является тип компонента-узла.

Парсер поддерживает таблицу именованных экземпляров узлов (конструкцию DEF/USE VRML, сходную с idref в XML), обеспечивая соответствующую семантику, а также считывание и создание ROUTE-объектов как адаптеров событий об изменениях значений свойств (PropertyChangeEvent adaptors).

Такая организация парсера обладает универсальностью (не пользуется ничем, кроме обычных средств манипулирования объектами и компонентами), является весьма компактной и достаточна для реализации представительного подмножества VRML вместе с расширенными (относительно VRML) средствами.

6. Модель и ее поведение

Для представления графа сцены (экземпляра 3D-модели) используется понятие контейнера, экземпляр которого может содержать в себе граф экземпляров JavaBeans-компонент, реализующих, в частности, узлы VRML. Мы используем две разновидности контейнеров (в свою очередь, являющихся JavaBeans-компонентами): SpaceContainer (определяет 3D-пространство сцены, наследуясь из бина Transform, реализующего одноименный узел VRML), и ActiveSpaceContainer (определяет 3D-пространство, добавляя возможность определять динамику поведения модели в отдельном потоке управления). В частности, при использовании потока управления для поддержки улов VRML TimeSensor, экземпляры которых находится в контейнере, реализуется стандартная модель поведения 3D-модели в зависимости от времени (моделируется 3D-пространство + время).

Как правило, реализации VRML-браузеров используют последовательную архитектуру, где есть цикл, состоящий из выполнения каскада событий, изначально вызванных срабатыванием сенсоров, имеющихся в сцене, соответствующего изменения состояния графа сцены и последующего траверсирования (возможно, в несколько приемов) графа для получения результата 3D-рендеринга. Каждый обход сопровождается использованием стека, где содержится необходимая контекстно-зависимая информация. При «повторном использовании» экземпляров узлов в разных ссылающихся на них элементах графа сцены (при присваивании в динамике одного экземпляра узла в качестве значения свойств других при изменениях графа, либо при обработке конструкции DEF/USE в парсере VRML) контекст использования влияет на отображаемый результат.

При реализации модели мы используем явное построение дерева контекстов для узлов графа. Наличие в модели собственно графа сцены сопровождается наличием в модели дерева контекстов. Дерево контекстов узлов строится вместе с графом узлов и поддерживается в соответствии с ним (*рис.1*). При этом значительно оптимизируется время, затрачиваемое на построение стеков и связанных с ними структур данных для хранения контекстно-зависимой информации при многократных траверсированиях графа. (Платой за заметное снижение издержек времени является некоторое увеличение издержек памяти, что оправдано для Java-реализации). Дерево контекстов имеет также естественную геометрическую интерпретацию: вертексы, составляющие треугольники, являются контекстами использования точек из определения этой поверхности (смежные треугольники «разделяют» общие точки).

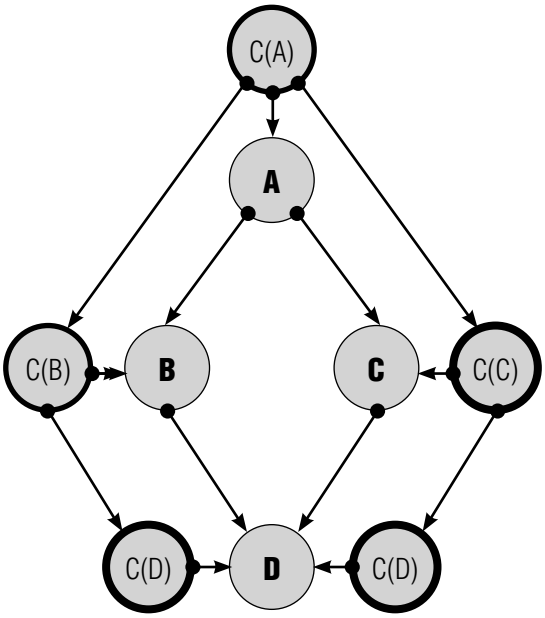

Рис. 1. Простейший граф узлов (DAG с узлами A, B, C, D) и дерево контекстов узлов.

Узел дерева контекстов узлов графа имеет ссылки на свой узел графа, а также на родительский и дочерние узлы данного дерева (если таковые есть). «Общий» узел D имеет два узла-контекста в дереве: один соответствует пути A-B-D, другой – пути A-C-D.

Для построения дерева контекстов по узлам графа классы реализации узлов графа предоставляют интерфейс создания своих контекстов. Каждый узел дерева контекстов связывает родительский и дочерние контексты дерева (последние всегда имеют ссылку на единственного родителя) и имеет ссылку на породивший его узел графа. Используемые ссылки являются «чувствительными» (sensitive references): если они ссылаются на объект, который может изменять свое состояние, они получают нотификации о соответствующих изменениях (участвуют в распространении событий PropertyChangedEvent, вызванных выполнением каскада событий в графе сцены).

Реализация объектов-контекстов для базовых компонент определяется их спецификой и обеспечивает хранение требуемой контекстно-зависимой информации. Например, компоненту Transform, реализующему узел VRML Transform, соответствует узел дерева контекстов типа TransformContext, хранящий значение матрицы преобразования координат для данного контекста относительно «мировой» системы координат контейнера сцены. Соответствующие типы контекстов предоставляются всеми компонентами, реализация поведения или отображения которых предполагает использование контекстно-зависимых данных (таких, которые иначе требуется постоянно вычислять и накапливать при траверсированиях графа). С другой стороны, контекстно-независимые компоненты (например, интерполяторы функций для анимаций) могут не участвовать в поддержке дерева контекстов. Наличие дерева контекстов для узлов графа позволяет контролировать выполнение требования ацикличности графа при его изменениях.

Поведение модели обеспечивается реализацией событийно-управляемых вычислений в графе. Первичным стимулом для распространения каскада событий является объект-сенсор, вырабатывающий исходное событие. Распространение события обеспечивается механизмом «связывания» свойств для JavaBeans-компонент, и выход из каскада событий обеспечивается правилами распространения событий PropertyChangeEvent (новое событие генерируется только при изменении значения свойства). Все JavaBeans-компоненты, реализующие узлы графа сцены, являются потокобезопасными (thread-safe); использованы synchronized-методы доступа к значениям свойств узлов. Объект модели может использоваться не только в разных контекстах одного контейнера, но и в разных контейнерах (в том числе – с разными потоками управления). Потокобезопасность компонент обеспечивает корректное поведение модели при всех обстоятельствах.

7. Организация отображения

Корнем дерева контекстов для графа сцены является корневой контекст ее контейнера. Это дерево описывает модель и поведение экземпляра контейнера, контролируемое графом распространения событий. Каждый экземпляр дерева контекстов может предоставлять свой вид (view) для последующего отображения (рендеринга). Таким образом, реализуется идеология model-view-controller (MVC) применительно к 3D-моделированию.

Для 3D-отображения на экране используется JavaBean-компонент Viewer, который представляет собой java.awt-компонент и может встраиваться в GUI java-приложений и апплетов. С каждым экземпляром Viewer'а связана камера, положение которой может определяться узлом Viewpoint (по правилам, обобщающим правила VRML, где отображение сцены параллельно с разных сторон не предусмотрено). Каждый Viewer имеет собственное независимое поведение (свой thread). Связывание модели с Viewer'ом происходит присваивани-

ем соответствующему свойству Viewer'а значения контейнера. Получив модель (контейнер), Viewer выполняет построение view модели в виде дерева объектов, корень которого определяется выбранным объектом Viewpoint (вычисления view выполняются относительно положения камеры, определяемого объектом Viewpoint).

Построение дерева view аналогично построению дерева контекстов для объектов в контейнере. Элементами дерева view являются экземпляры классов-view, инстанциируемые соответствующими классами-контекстами. Подобно тому, как контексты экземпляров компонент, реализующих граф сцены, являются организованными в дерево ссылками на узлы графа, элементами дерева view являются ссылки на элементы дерева контекстов. В реализации использовано соглашение об именовании классов: если Node – это тип компонента для узла графа, то класс NodeContext определяет реализацию элементов дерева контекстов, а NodeContextView – реализацию view для соответствующего типа контекста.

Поведение каждого Viewer'а является потребителем данных из дерева контекстов, тогда как поведение модели (графа сцены) является поставщиком новых данных. В соответствии с этим организуется синхронизация потоков управления.

JavaBean-компонент Viewer обеспечивает 3D-рендеринг на своем дереве view при изменениях, о которых сигнализируют элементы дерева контекстов, оповещаемые об изменениях узлами графа. Компонент Viewer обеспечивает отображение на экран полностью программными средствами Java (100% Java – код). Набор JavaBean-компонент, способных отображать поведение модели, не ограничивается такой программной реализацией. В частности, имеется компонент, просто формирующий изображение в памяти (невидимый компонент-Viewer). Результирующее изображение может использоваться в качестве текстуры в самой сцене. Компонентность позволяет легко пополнять номенклатуру компонент-Viewer'ов, добавляя компоненты с различной реализацией рендеринга (например с применением OpenGL, DirectX или иных средств визуализации). Компонентный подход позволяет организовать различные сценарии использования имеющихся компонент и интерактивно реализовать эти сценарии в контейнере JavaBeans-компонент.

8. Примеры

Разработанная библиотека JavaBeans-компонент позволяет строить 3D-модели непосредственно в стандартном BeanBox-контейнере. На *рис. 2* показан BeanBox, в котором из имеющихся компонент, представленных на левой панели, собран и отображен простейший 3D-объект – треугольник. Все значения свойств использованных объектов настроены с помощью стандартных манипуляций с (правой) панелью свойств. Для визуализации инстанциирован компонент Viewer. При привязке к контейнеру модели он показывает 3D-изображение. Контейнер модели с ее поведением можно сравнить с телестудией, где происходит действие. Viewer играет роль телевизора, который надо включить и связать с имеющейся студией, чтобы увидеть происходящее действие.

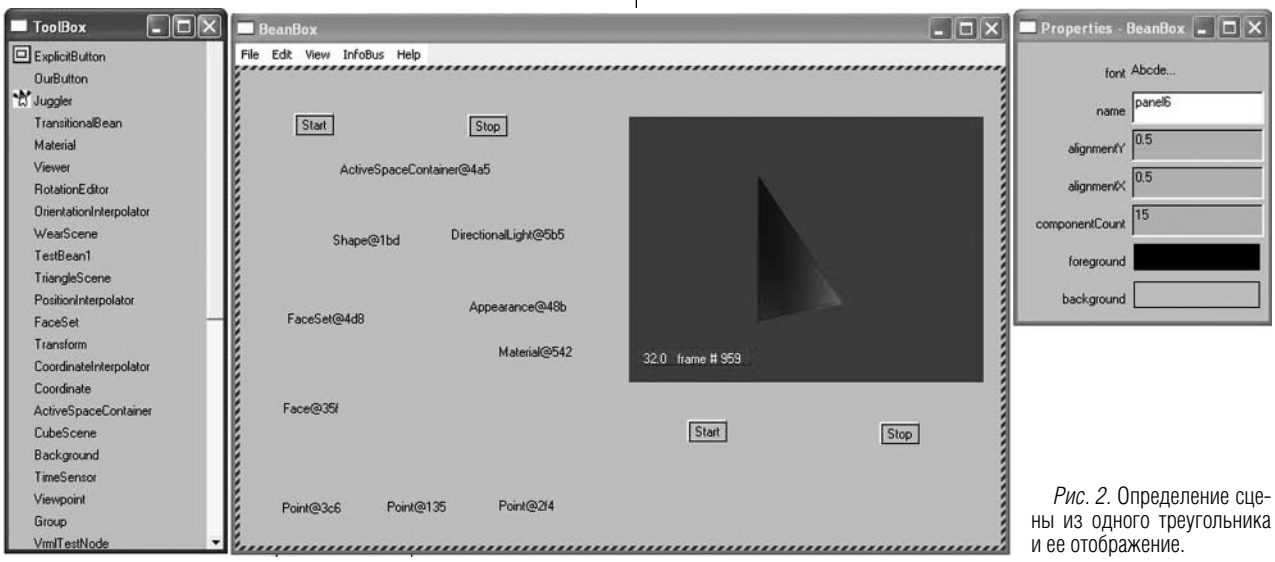

Рис. 2. Определение сцены из одного треугольника и ее отображение.

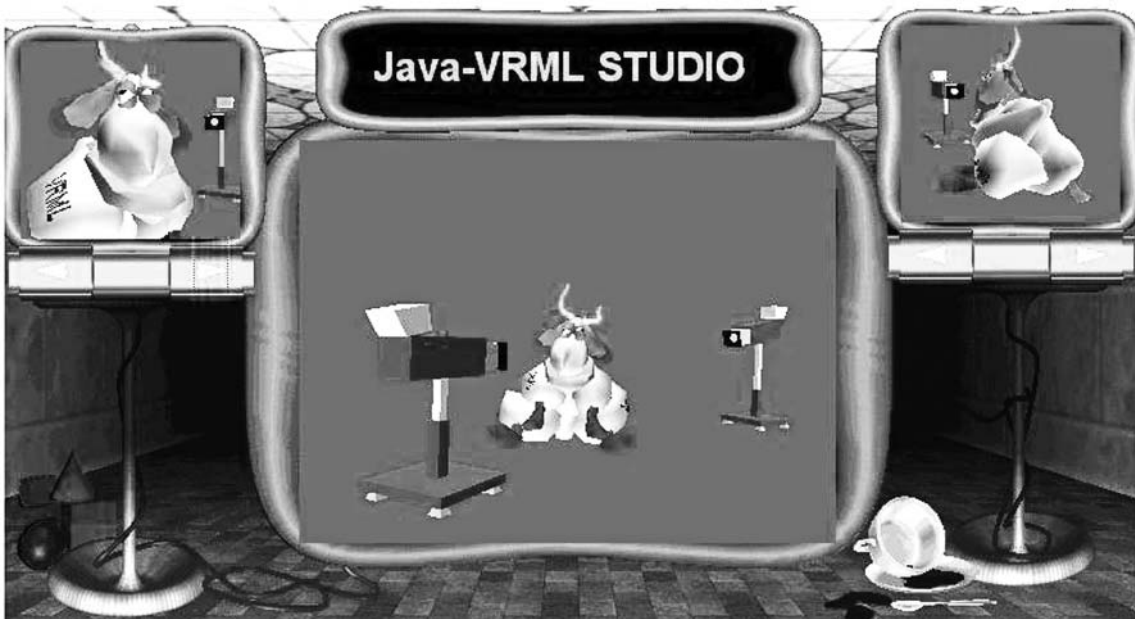

Рис.3. Сцена с тремя экранами: общий вид и два изображения от камер в сцене.

Продолжая эту аналогию, рассмотрим использование компонент, показанное на *рис. 3*. Здесь показан Java-апплет, в котором размещены три экземпляра компонента-Viewer'а. Модель представляет собой персонаж и две телекамеры, которые можно перемещать по кругу нажатием кнопок под левым и правым экранами. Каждый экран показывает изображение с соответствующей ему телекамеры. Центральный экран показывает поведение всей сцены третьей камерой, которую можно увидеть на боковых экранах при надлежащем расположении их камер. Этот пример иллюстрирует возможности параллельного отображения одной модели. Все три экземпляра Viewer'а работают параллельно друг с другом и с поведением модели (персонажа и камер).

Следующий пример на *рис. 4* показывает BeanBox, в котором имеется три экземпляра ActiveSpaceContainer'а.

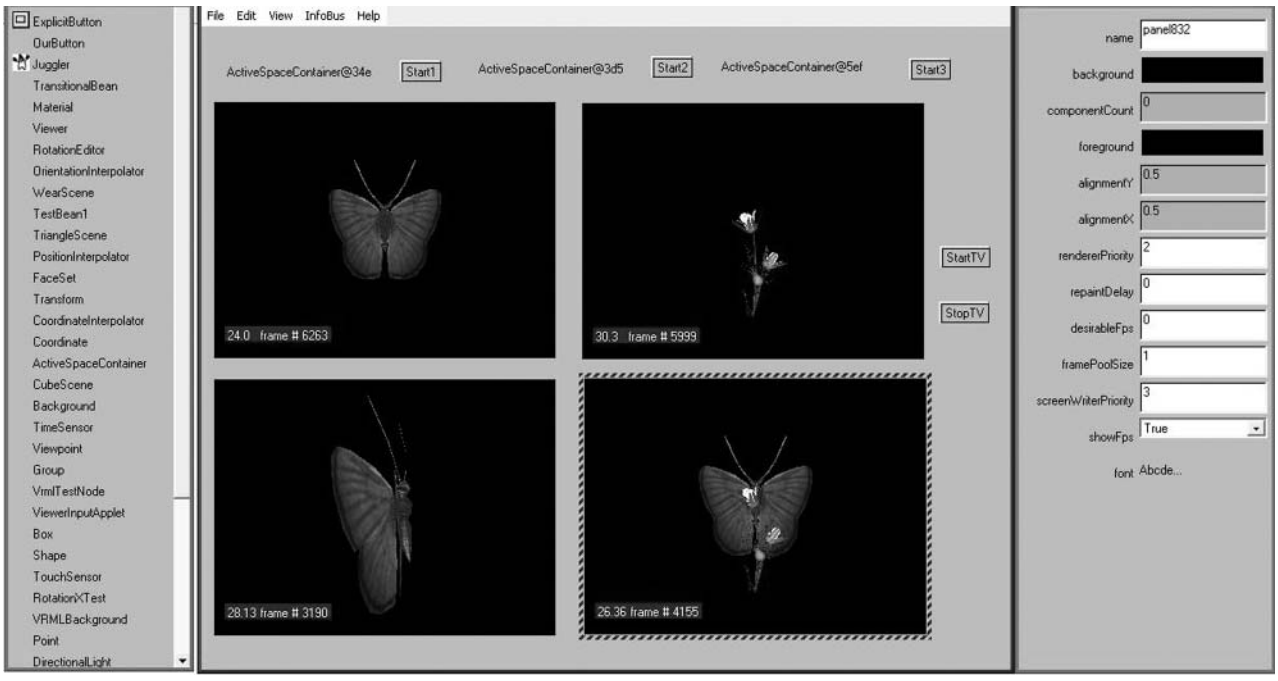

Рис. 4. Параллельное поведение и отображение 3D-моделей.

В первый экземпляр контейнера загружен VRML-файл с описанием геометрии и поведения бабочки. Этот контейнер сопоставлен с двумя экземплярами Viewer'а (расположенными слева), которые параллельно отображают поведение 3D-модели бабочки с разных ракурсов (Viewpoint'ов). Второй экземпляр контейнера содержит загруженную из VRML-файла модель цветка, которая обладает собственным поведением. Это поведение модели отображается параллельно в правом верхнем экземпляре Viewer'а. Наконец, показанный третий экземпляр контейнера содержит в себе первый и второй контейнеры (располагая ссылками на них). Поведение этого контейнера является совокупным поведением первых двух, и мы наблюдаем поведение этой составной модели в четвертом (правом нижнем) экземпляре Viewer'а. Все необходимые для этого построения были произведены визуально и интерактивно стандартными средствами BeanBox-контейнера.

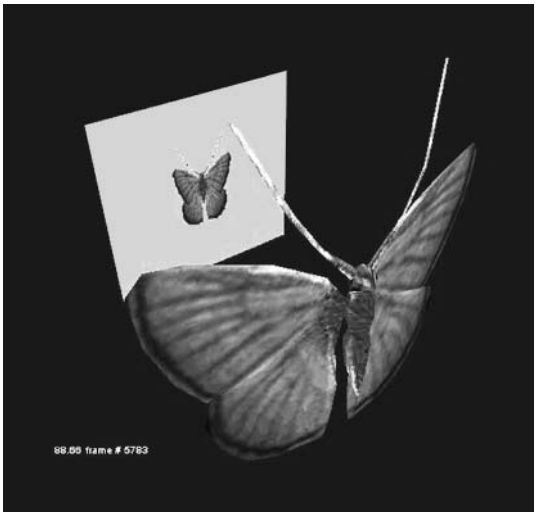

Рис. 5. Рендеринг для получения анимированной текстуры и общее отображение.

На *Рис. 5* показан Java-апплет с поведением бабочки. Результат визуализации, получаемый экземпляром компонента-Viewer'а в памяти, используется в качестве (анимированной) текстуры для прямоугольной поверхности. Модель, состоящая из бабочки и этой текстурированной поверхности, отображается в апплете другим экземпляром Viewer'а.

9. Заключение

Компонентно-ориентированное программирование является передовым направлением разработки программного обеспечения [32, 33, 34]. Развитие средств компонентного программирования позволит перейти от построчного кодирования программ к их промышленному производству и созданию новых рынков программных компонент, подобно тому, как это имело место при производстве аппаратных компонент.

При компонентно-ориентированной разработке существуют два основных вида деятельности. Первый – разработка собственно повторно используемых компонент. Второй – разработка программного обеспечения с помощью существующих компонент. Такой подход соответствует идеологии компонентной модели JavaBeans. Однако, данная компонентная модель не обеспечивает должным образом построения самих компонент из предварительно подготовленных: новые компоненты появляются в результате компиляции или кодогенерации. BDK не позволяет пополнить номенклатуру компонент, не прибегая к загрузке соответствующих Java-классов реализации.

С другой стороны, язык VRML предполагает наличие средств определения новых типов узлов-компонент с помощью конструкций PROTO (EXTERNPROTO). Семантика этих конструкций, однако, не отвечает принципам (class-based) объектно-ориентированного программирования. В литературе обсуждались попытки совершенствования языка VRML в этом направлении [35, 36], однако эти обсуждения не нашли надлежащего воплощения. По этой причине мы пока исключили реализацию декларативных средств описания определяемых типов из подмножества средств VRML, реализованных с применением JavaBeans-компонент в данной работе.

Для того чтобы реализовать возможность динамического (без Java-компиляции или генерации Java-байт-кодов) определения составных типов компонент с использованием базового набора JavaBeans-компонент, требуется расширение компонентной модели JavaBeans в направлении больших возможностей композиции/декомпозиции компонент и соответствующие расширения возможностей контейнеров и инструментов композиции. Разработка и реализация соответствующих средств составляют предмет дальнейших исследований. ■

10. Литература

1. TIOBE Programming Community Index for June 2010. [Электронный ресурс]. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
2. Java3D. [Электронный ресурс]. URL: https://java3d.dev.java.net/
3. ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2004 — Virtual Reality Modeling Language (VRML). [Электронный ресурс]. URL: http://www.web3d.org/x3d/specifications/vrml/
4. Sun Microsystems. JavaBeans Specification v1.0.1 , July 1997. [Электронный ресурс]. URL: http://java.sun.com/products/javabeans/docs/spec.html,http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html
5. Bean Development Kit. См. http://life.csu.edu.au/java-tut/javabeans/index.html
6. Спецификации X3D. См. http://www.web3d.org/x3d/specifications/
7. XML. [Электронный ресурс]. URL: http://www.w3.org/XML/
8. XUL. [Электронный ресурс]. URL: https://developer.mozilla.org/
9. XAML. [Электронный ресурс]. URL: http://windowsclient.net/
10. BeanBuilder. [Электронный ресурс]. URL: https://bean-builder.dev.java.net/
11. B.P. Zeigler. Theory of Modeling and Simulation. 1976.
12. DEVSJAVA. [Электронный ресурс]. URL: http://www.acims.arizona.edu/SOFTWARE/software.shtml#DEVSJAVA
13. PtolemyII. Heterogeneus concurrent modeling and design in Java. Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. v. 1,2,3. [Электронный ресурс]. URL: http://ptolemy.eecs.berkeley.edu
14. OMNeT++ Community Site. См. www.omnetpp.org
15. Bruneton E., Coupaye T., Stefani J.B. The Fractal Component Model specification. Version 2.0-3. The ObjectWeb Consortium, 2004.
16. The fractal Component Model. ObjectWeb Consortium. См. http://fractal.objectweb.org
17. Herbert Praehofer, Johannes Sametinger, Alois Stritzinger. Discrete Event Simulation using the JavaBeans Component Model.
18. Herbert Praehofer, Johannes Sametinger, Alois Stritzinger. Concepts and Architecture of a Simulation Framework Based on the JavaBeans Component Model
19. Mark Pesce. VRML-Browsing and Building Cyberspace, New Riders Publishing, 1995.
20. Xj3D. [Электронный ресурс]. URL: http://www.xj3d.org/
21. Cortona 3D. [Электронный ресурс]. URL: http://www.cortona3d.com/Home.aspx
22. BS Contact J. [Электронный ресурс]. URL: http://www.bitmanagement.de/en/products/interactive-3d-clients/bs-contact-j
23. Demicron WireFusion. [Электронный ресурс]. URL: http://www.demicron.com/
24. JavaFX. [Электронный ресурс]. URL: http://www.javafx.com/
25. Weaver, James L., JavaFX Script: Dynamic Java Scripting for Rich Internet/Client-side Applications, Apress, 2007.
26. Microsoft Silverlight. [Электронный ресурс]. URL: http://www.microsoft.com/silverlight/
27. 3D with JavaFX. [Электронный ресурс]. URL: http://learnjavafx.typepad.com/weblog/3d-with-javafx/
28. 3DMLW, 3D Technologies R&D [Электронный ресурс]. URL: http://www.3dmlw.com/
29. COLLADA. [Электронный ресурс]. URL: http://www.khronos.org/collada/
30. Universal 3D File Format. [Электронный ресурс]. Систем. требования: Adobe Acrobat Reader. – URL: http://www.ecma-international.org/activities/Communications/U3D-presentation-public.pdf . URL: http://meshlab.sourceforge.net/wiki/images/c/cc/Laurana.pdf
31. Long Term Persistence of JavaBeans Components: XML Schema. [Электронный ресурс]. URL: http://java.sun.com/products/jfc/tsc/articles/persistence3/
32. Szyperski, C., Component Software – Beyond Object-Oriented Programming. Reading, Massachusetts: Addison-Wesley, 1998.
33. Clemens Szyperski,Dominik Gruntz,Stephan Murer. Component software: beyond object-oriented programming. ACM Press, Second Edition, 2002.
34. Wang, Andy Ju An., Component-oriented programming / Andy Ju An Wang & Kai Qian. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.
35. Curtis Beeson. An Object-Oriented Approach To VRML Development. Silicon Graphics Inc.
36. Stephan Diehl. VRML++: A Language for Object-Oriented Virtual-RealityModels, Proceedings of the 24th International Conference on Technology of Object-Oriented Languages and Systems TOOLS Asia, Bejing, 1997.