электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.415.538

Описание работы основных механизмов программной системы 3D визуализации с использованием визуального языка OBeML

Гергиева И.Р., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Информационные системы и телекоммуникации»

Научный руководитель: Иванов А.М., старший преподаватель
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
deviatkov@bmstu.ru

1 Введение и проблема

Программные системы с большим количеством кода сложны, поэтому для описания механизма исполнения программы используются диаграммы разных типов, описывающие разные аспекты работы механизма, что не позволяет быстро получить целостное представление о работе программы. Существует множество инструментов для создания графического описания структуры и поведений сложных систем, использующих нотацию UML[3]. Примером может служить плагин JIVE[2] для платформы Eclipse. JIVE – среда визуальной отладки программ, написанных на Java. Он отражает работу программных систем с помощью расширенной объектной диаграммы, в состав которых входят диаграммы формулировки запросов для исполнения других программ, диаграммы состояния времени исполнения, диаграммы трассировки исполнения программы, диаграммы с отображением последовательности вызовов в программе, диаграммы, отражающие - какие компоненты программы вызывают друг друга.

В настоящее время UML является языком моделирования, наиболее часто используемым для анализа, описания и разработки системы программного обеспечения[1]. Поэтому любые новые нотации и языки стоит сравнивать с UML.

Для анализа простоты использования языка UML была проведена обратная разработка 3D приложения на основе графа сцены jME в среде разработки и моделирования RSA (Rational Software Architect)[4]. Описание на языке UML представляет собой диаграммы разных типов: диаграммы классов, диаграммы объектов и диаграммы поведений. Для программиста такое разделение диаграмм на несколько видов существенно усложняет процесс понимания работы как большого приложения в целом, так и работы отдельных его частей. Это связанно с тем, что довольно сложно удержать в

голове одновременно все эти диаграммы и связи, отраженные на них. Поэтому для упрощения задачи понимания был разработан визуальный язык OBeML, дающий возможность изобразить все виды диаграмм, описанных на UML, на одной диаграмме описания работы приложения. Этот факт дает возможность программисту видеть одновременно все аспекты работы механизмов.

2 Описание работы приложения aircarrier на языке UML

Работа приложения *aircarrier,* реализованного на графическом каркасе *JMonkeyEngine* представлена на нескольких диаграммах, созданных с помощью UML. На рисунке 3-1 представлена диаграмма классов, созданная в среде разработки и моделирования RSA (Rational Software Architect), использующей UML. На ней показаны классы, их наследники и связи между классами: наследование, реализация и зависимость.

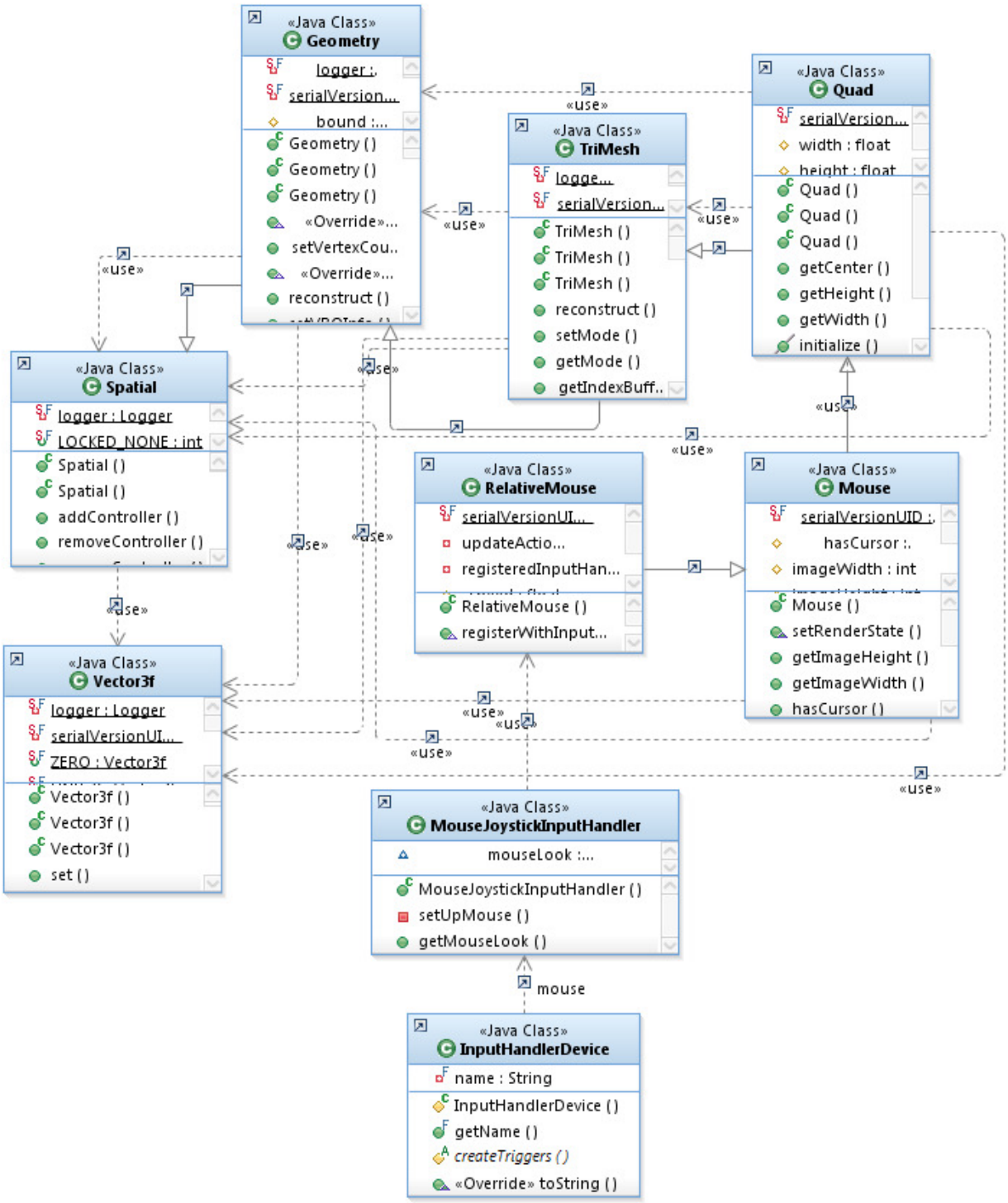


Рис. 3-1. Диаграмма классов на UML

На рисунке 3-2 приведена объектная диаграмма, также созданная в среде RSA на UML.

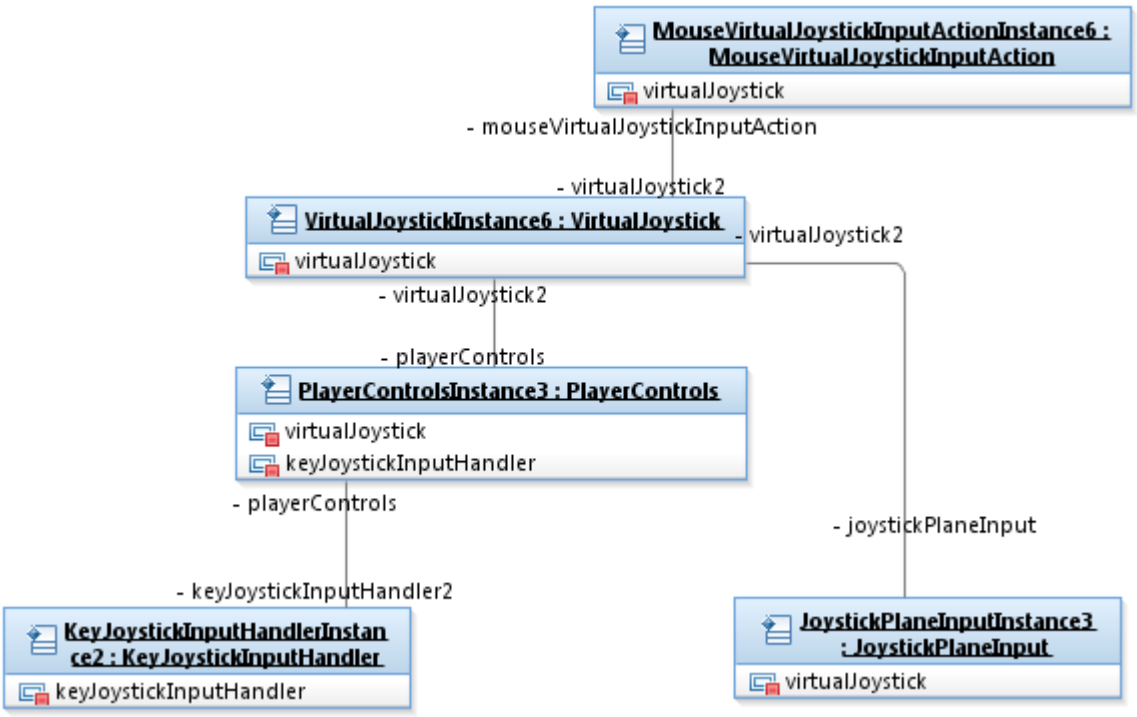


Рис. 3-2 Объектная диаграмма на UML

На ней изображены объекты классов, атрибуты этих объектов и связи между объектами, но нет на этой диаграмме возможности показать наследование классов, что затрудняет процесс понимания структуры программы.

Далее на рисунке 3-3 изображена диаграмма последовательности вызовов методов также на UML.

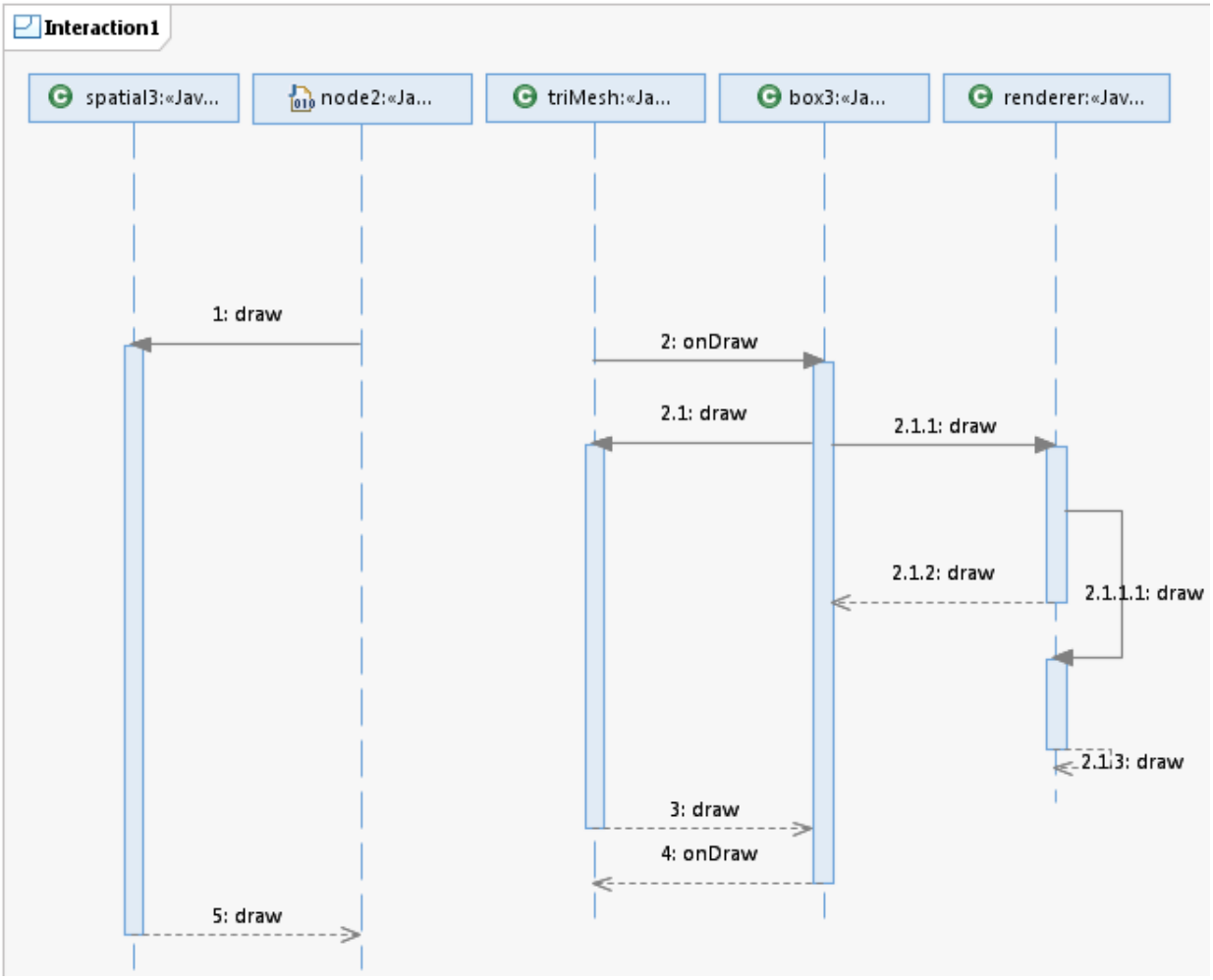


Рис. 3-3. Диаграмма последовательности вызовов на UML

На этой диаграмме также нет возможности показать наследование между классами, как и на объектной диаграмме, представленной на рисунке 3-2, и нет возможности установить связи между объектами и соответственно связи между классами, на ней показана только последовательность вызовов определенных методов из определенных классов.

В результате программисту необходимо создавать, одновременно читать и анализировать сразу три диаграммы, это неудобно и трудно для понимания того, как работают отдельные структуры большой программы.

3 Визуальный язык OBeML для описания работы механизмов программных систем

Для описания работы приложений предлагается визуальный язык OBeML (Object Behavior Modeling Language).

Основные особенности языка: всего один тип диаграммы, визуализация одной только структуры объектов программного механизма в конкретный момент его

жизненного цикла и взаимодействия этих объектов на одной диаграмме. При визуализации структуры объектов используется необходимая и релевантная информация из структуры классов.

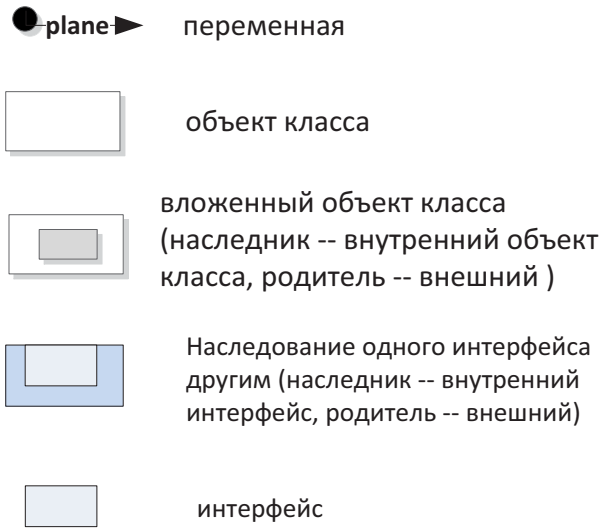
Фрагмент грамматики визуального языка OBeML представлен в нотации EBNF [5], состоящий из основных и дополнительных элементов и правил. Основные и дополнительные элементы грамматики: объект, переменная, метод, наследник, интерфейс, константа, имя_класса.

Правила в соответствии с перечисленными элементами примут вид:

объект = {переменная | метод | наследник};

интерфейс = {наследник | имя_класса | метод | константа}.

Графические обозначения элементов визуального языка OBeML:



4 Пример описания работы фрагмента 3D приложения на языке OBeML

На рисунке 3-4 представлен фрагмент диаграммы описания работы приложения *aircarrier* на основе графического каркаса *JMonkeyEngine* на языке OBeML.

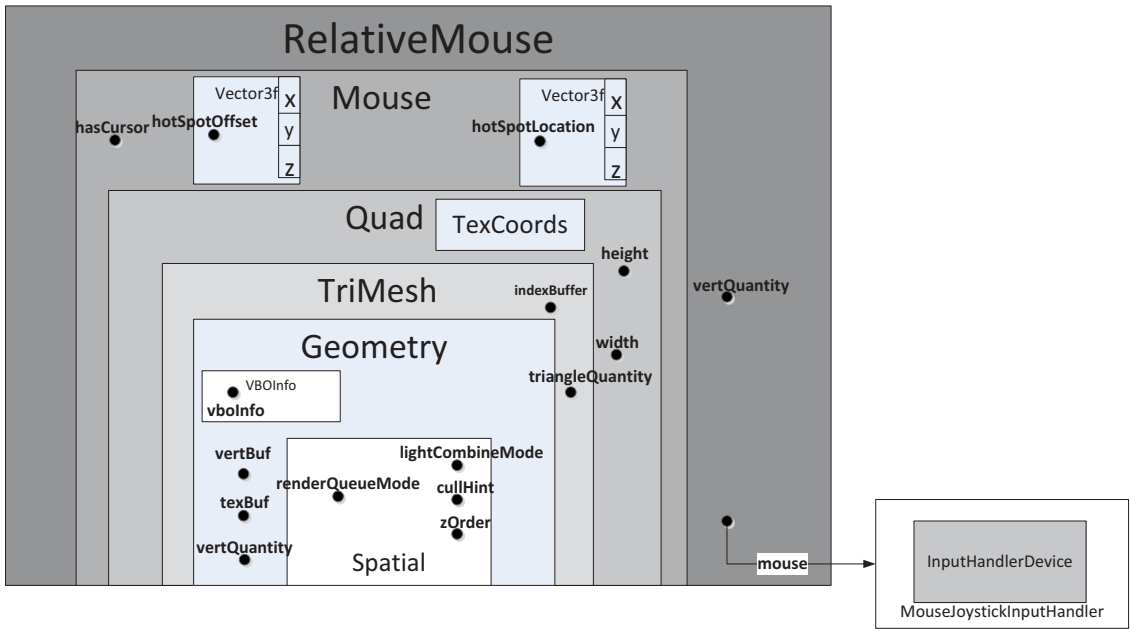


Рис. 3-4. Диаграмма работы приложения aircarrier, описывающая наследования и отображающая обозначения переменных на ObeML

На диаграмме видно, что классы *RelativeMouse*, *Mouse, Quad, TriMesh, Geometry* являются наследниками класса *Spatial*. Класс *Vector3f* является наследником класса *Mouse*, в нем инициализируются и пересчитываются координаты (x, y, z) положения объекта в момент его движения. На диаграммах классов на UML наследование можно показать только на таком виде диаграмм – на диаграммах классов (рисунок 3-1).

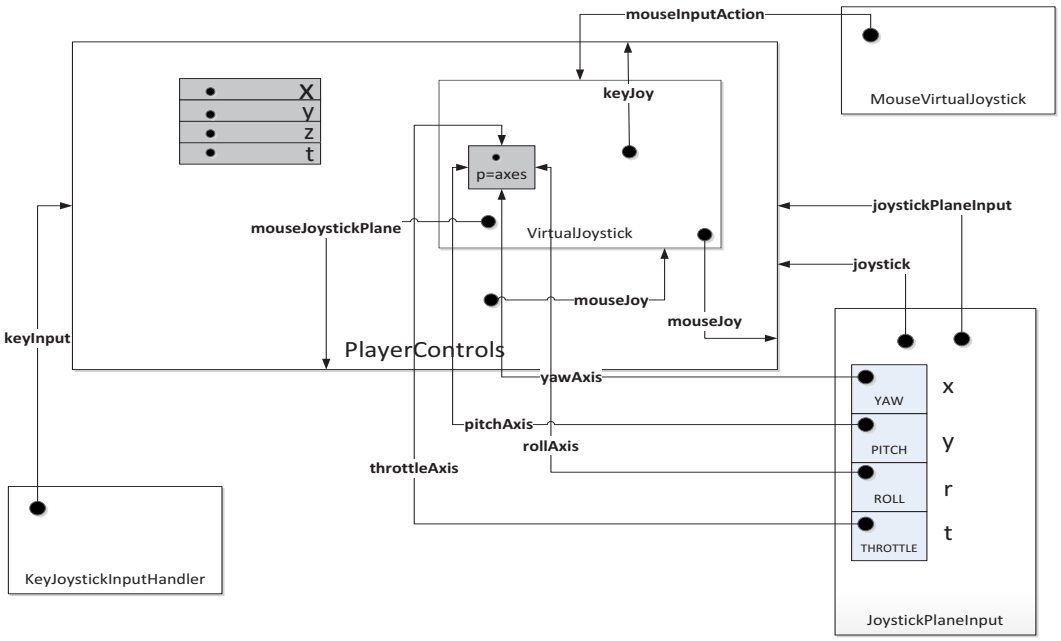


Рис. 3-5. Диаграмма работы приложения aircarrier, описывающее движение данных к виртуальному джойстику на OBeML

На рисунке 3-5 представлен кусок из объектной диаграммы описания работы приложения *aircarrier* на OBeML, отражающий поступление данных от реальных устройств управления в виртуальный джойстик и сбор этих данных в классе *VirtualJoystick*.

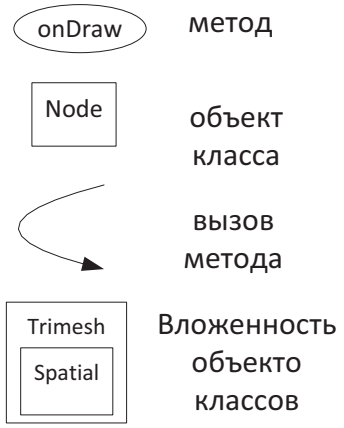
В класс *VirtualJoystick* информация поступает через объекты класса *PlayerControls, KeyJoystickInputHandler*, *JoystickPlaneInput, MouseVirtualJoystick.* По переменным *mouseJoy, joystickPlaneInput, keyInput, joystick* с помощью объектов этих классов в класс *VirtualJoystick* поступает информация о том, какая клавиша нажата K, L, J в данный момент. Вызовом методов *setAxisValue* и *getAxisValue* устанавливаем и получаем значение переменной *axisChanged* класса *VirtualJoystick* для каждого прохода по значениям координат (x, y, z), определяющих расположение осей, т.е. изменение положения осей, относительно движения трехмерной модели объекта. От объектов класса *JoystickPlaneInput* в класс *VirtualJoystick* поступает информация о положении объекта по переменным *yawAxis, pitchAxis, rollAxis, throttleAxis* (рисунок 3-5).

Вся информация о положении мышки, нажатой кнопки или нажатой клавиши собирается в классах *RelativeMouse, PlayerContlos, MouseVirtualJoystickInputAction*. Информация о положении трехмерной модели относительно осей (переменные *yawAxis, rollAxis, throttleAxis, pitchAxis*) из объектов класса *JoystickPlaneInput* передается в класс *VirtualJoystick*.

Координаты положения объекта хранятся и пересчитываются при поступлении новой информации от устройств управления в объектах класса *Vector3f* и также передаются в класс *MouseJoystickInputHandler,* и далее в класс *VirtualJoystick* для виртуального управления объектом трехмерной модели.

Также можно описать в виде диаграмм процесс рендеринга трехмерной модели. На рисунке 3-6 представлено описание рендеринга объекта трехмерной модели на OBeML.

Принятые условные обозначения:



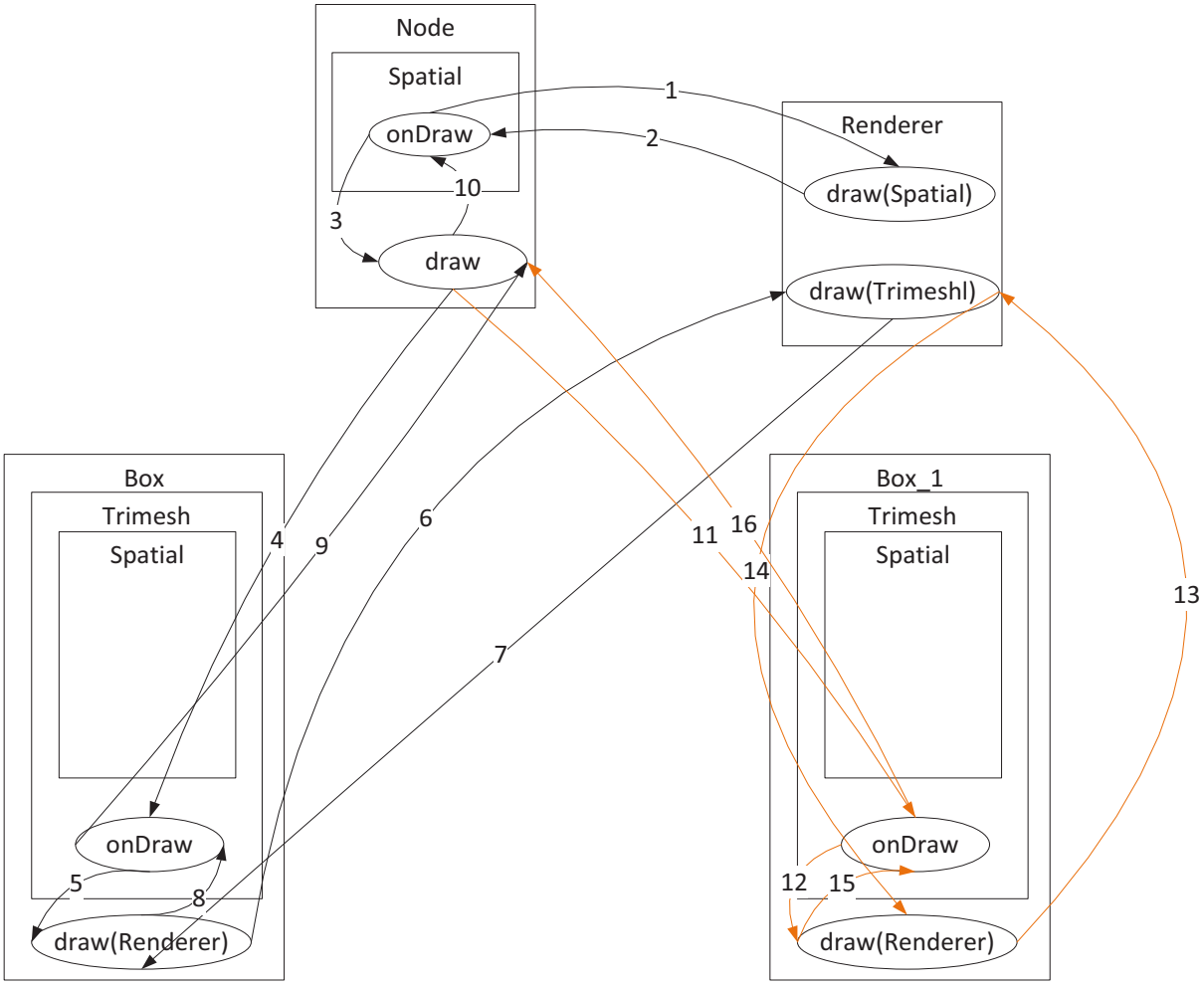


Рис. 3-6. Диаграмма описания рендеринга трехмерной модели на ObeML

На данной диаграмме показаны вложенность объектов и какие именно методы и в какой последовательности вызываются в процессе рендеринга. Так, например, вызывается метод *onDraw* класса *Spatial*; далее вызывается метод *draw* класса *Renderer*, работающий с объектом класса *Spatial*; затем вызывается метод *draw* класса *Node*; далее метод *onDraw* класса *Trimesh*, а класс *Trimesh* является родителем класса *Box*, т е уже рендеринг объекта модели; далее вызывается метод *draw* в классе *Box*, работающий с объектом класса *Renderer*. Потом возвращаемся в класс *Renderer*, в нем вызывается метод *draw*, работающий с объектом класса *Trimesh*. Аналогичные вызовы методов в классах происходят для рендеринга следующего объекта модели. В результате на диаграмме помимо последовательности вызовов методов отражены наследование классов, объектов.

5 Выводы

Предложенное описание механизмов работы программной системы на визуальном языке OBeML дает возможность увидеть все необходимые виды структур программы

(структуру классов, структуру объектов и поведений), а также отобразить отношения элементов структур между собой одновременно на одной диаграмме. Это позволяет лучше и быстрее понять как отдельные механизмы работы объемных программных систем, так и работу приложений в целом.

Список литературы

1. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон Объектно-ориентированный анализ и проектирование с примерами приложений. М.: Вильямс, 2008. – 720с.
2. Bial Alsallakh, Peter Bodesinsky, Alexander Gruber, Silvia Miksch. “Visual Tracing for the Eclipse Java Debugger”. Center of Visual Analytics Science and Technology (CVAST) Institute of Software Technology and Interactive Systems Vienna University of Tachnology Vienna, Austria.
3. ISO/IEC 14977:1996 – Information technology – Semantic metalanguage – Extended BNF.