РЕАЛИЗАЦИЯ СИСТЕМЫ АГЕНТНОГО ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ В WEB

А. Ю. Гарбарев (Омск)

Введение

В настоящий момент существует большое количество систем имитационного моделирования, каждая из которых имеет свои преимущества и недостатки. Так, на кафедре АСОИУ Омского государственного технического университета существует лаборатория имитационного моделирования imlab, в которой разработаны такие системы, как Simulab и SIMBIGRAPH. Данные системы имеют ряд преимуществ по сравнению с существующими аналогами (AnyLogic, RepastS, ASCAPE, NetLogo) и вполне могут конкурировать на рынке имитационных систем. Однако развитие современного сетевого общества требует разработки веб-ориентированных интерфейсов для вовлечения как можно большего числа пользователей. В докладе обсуждается клиент-серверная система агентного моделирования BridgeGL, позволяющая реализовать функционал SIMBIGRAPH в веб.

Архитектура системы

Разработанная система состоит из трех частей: клиентское приложение, серверное приложение и подсистема моделирования (рис. 1). Клиентское приложение отвечает за взаимодействие с пользователем, а также за отображение процесса моделирования в браузере. Серверная часть отвечает за взаимодействие клиентской части с подсистемой моделирования. Подсистема моделирования выполняет модель, загруженную пользователем.

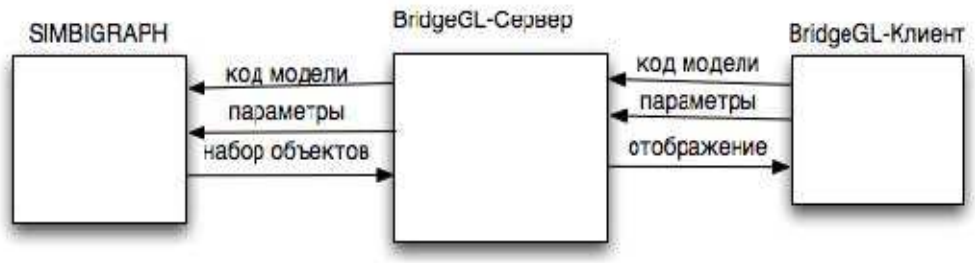

Рис. 1. Архитектура системы

Серверная часть

Серверная часть реализована на языке Java и представляет собой приложение с использованием сервлетов и страниц JSP. Серверная часть обеспечивает взаимодействие клиентской части с подсистемой моделирования, в частности управляет процессом моделирования, выполняет загрузку модели, предоставляет клиентской части состояние модели в текстовом формате.

Серверное приложение (рис. 2) состоит из подсистемы моделирования Simulation, пакета BridgeGL-Server и пакета BridgeGL. Когда пользователь выбирает агентную модель, пакет BridgeGL-Server инициализирует модель и запускает процесс моделирования. Запущенный процесс моделирования выполняется с помощью подсистемы моделирования Simulation. Пакет BridgeGL, получая объекты от подсистемы моделирования, преобразует их в текстовый формат, готовый для отображения на клиентской части. Затем, с определенным интервалом, пакет BridgeGL передает текстовое представле-

ние сцены в пакет BridgeGL-Server, после чего эти данные отправляются на клиентскую часть.

Клиентская часть

Клиентская часть представляет собой JavaScript приложение, которое с помощью технологии WebGL и библиотеки SceneJS отображает полученное с серверной части состояние модели в окне браузера.

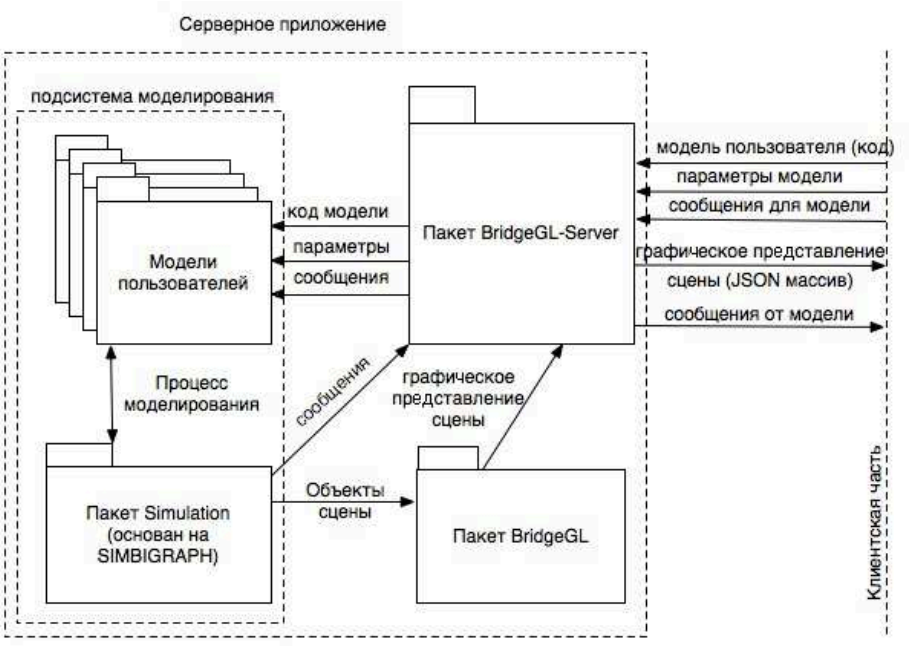

Рис. 2. Структура серверной части

Использование системы и направления развития

На рис. 3 представлен интерфейс пользовательской части. Клиентская часть позволяет работать с моделями, загруженными на сервер. В процессе работы пользователь может пошагово наблюдать процесс моделирования, изменять шаг и отслеживать информацию о текущем состоянии системы.

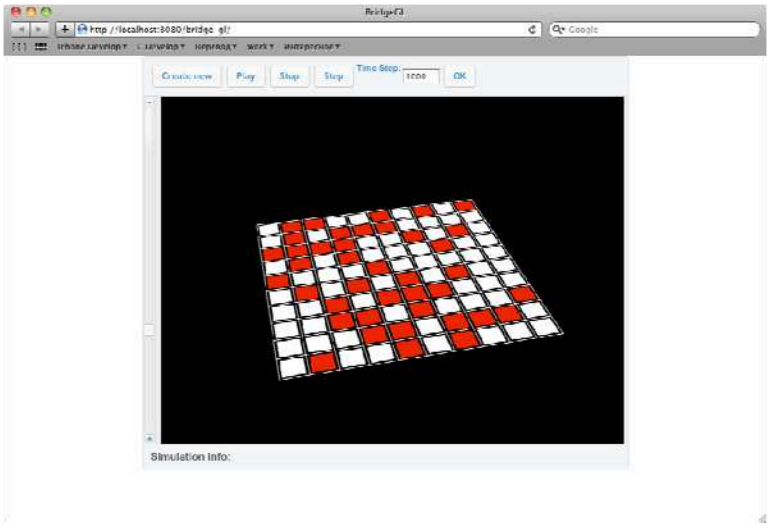

Рис. 3. Пользовательский интерфейс системы Bridge-GL

Рассмотрим программный интерфейс, позволяющий пользователям создавать модели. Пользовательская модель представляет собой исходный код, написанный на языке Java. Каждая модель должна наследоваться от класса Simulation, а значит, реализовывать методы: initScene(ArrayList args), performStep(), getStatistics().

Реализация игры «Жизнь» в BridgeGL показана на рис. 4.

```
public class GameOfLife extends Simulation {

    //Агента наследуем от BGLAgent
    //для того что бы он мог отображаться
    class Cell extends BGLAgent {

        //Реализация методов setState, getState
        // и storeState тривиальна

        public void step() {

            //Получаем соседей по Муру
            ArrayList neighbors = grid.getMooreNeighbors(this);

            //...
            //Считаем кол-во живых из соседей
            //...

            //2 правила игры "жизнь"
            if (state == AgentState.DEAD && count_of_live == 3) {
                setState(AgentState.LIFE);
                return;
            }
            if (state == AgentState.LIFE &&
                (count_of_live < 2 || count_of_live > 3)) {
                setState(AgentState.DEAD);
                return;
            }
        }
    }

    @Override
    public void initScene(ArrayList args) {

        //Создаем объект "решетка" размером 20 на 20
        grid = new BGLGrid(20, 1, 20, 5, 0, 5);

        scene.addNode(grid);

        //Добавляем агентов на решетку
        doStructure();
    }

    @Override
    public void performStep() {

        for (int i = 0; i < grid.size_x; i++)
            for (int j = 0; j < grid.size_z; j++)
                ((Cell) grid.getAt(i, 0, j)).storeState();

        for (int i = 0; i < grid.size_x; i++)
            for (int j = 0; j < grid.size_z; j++)
                ((Cell) grid.getAt(i, 0, j)).step();
    }

    @Override
    public String getStatistics() {

        //...
        //Получаем текущий шаг и кол-во живых клеток
        //
        return String.format("Шаг: %d Кол-во живых: %d", currentStep++, count);
    }
}
```

Рис. 4. Пример реализации игры «Жизнь Конвея»

Ключевым объектом в системе является объект *scene*. Любой объект, добавленный к объекту *scene* с помощью *addNode*, становится видимым пользователю. Цель метода *initScene* – создать объекты, задающие начальное положение модели, и добавить их в объект *scene*. Например, в реализации на рис. 4 создается объект решетки и добавляется на сцену. Рассмотрим задачу остальных методов: *getStatistics* – возвращает строку, которая будет отображена пользователю; *performStep* – вызывается с указанным

пользователем шагом и его цель – это изменение состояния модели, т.е. процесс моделирования. Класс *Cell* исполняет роль агента и имеет метод *step*, который, как видно в реализации, вызывает каждый шаг моделирования у каждого агента, что задает поведение агентов.

Как видно, создание модели максимально упрощено, что позволяет пользователям сосредоточиться на логике модели, а не на программной реализации.

В процессе эксплуатации системы выявлено узкое место в производительности системы – передача данных от сервера клиенту. Так, при работе с моделями, где количество агентов более 1600, быстродействие клиентской части снижалось. Это обусловлено обработкой сообщений большого объема. Для увеличения производительности в данный момент происходит портирование подсистемы моделирования на клиентское приложение, таким образом, при анимации не тратится время на формирование, передачу и обработку сообщений о каждом состоянии модели. Для получения непосредственного доступа к видеокарте решено отказаться от вспомогательных библиотек, таких как SceneJS, и работать напрямую с WebGL. В данный момент помимо реализации системы, обеспечивающей исполнение процесса моделирования на сервере, существуют наработки, где моделирование выполняется на стороне клиента [6].

Заключение

Разработана система агентного моделирования BridgeGL, обеспечивающая:

- загрузку и проведение имитационного моделирования удаленно (на сервере);
- визуализацию модели на стороне клиента (в браузере);
- эффективную реализацию моделирования клеточных автоматов.

На стороне сервера BridgeGL использует JavaEE технологии, на стороне клиента – технологии WebGL и SceneJS, взаимодействие клиент–сервер осуществляется посредством JSON массивов. В настоящее время ведется усовершенствование подсистемы, размещенной на стороне клиента, направленное на существенное увеличение производительности путем переноса части вычислений на сторону клиента [6].

Литература

1. **Ершов Е. С., Юдин Е. Б** Система имитационного моделирования SIMULAB // Имитационное моделирование. Теория и практика (ИММОД-2009): Материалы 3 й всероссийской конференции. Т. 1. СПб: ЦТСС, 2009. С. 257–261.
2. **Юдин Е. Б.** Система агентного моделирования Simbigraph // Россия молодая: передовые технологии – в промышленность: Матер. III Всерос. науч.-техн. конф. Омск: Изд-во ОмГТУ, 2010. Кн. 1. С. 312–316.
3. Сообщество WebGL. [Электронный ресурс] / Режим доступа: http://learningwebgl.com, свободный. Яз. англ.
4. Черновая спецификация WebGL [Электронный ресурс] / Режим доступа: https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/doc/spec/WebGL-spec.html, свободный. Яз. англ
5. Библиотека SceneJS [Электронный ресурс] / Режим доступа: http://www.scenejs.org/, свободный. Яз. англ.
6. **Гарбарев А. Ю.** Демонстрационные примеры – вычисление на клиентской части [Электронный ресурс] / Режим доступа: http://alexgarbarev.homeunix.org/modeling