# СИСТЕМА АГЕНТНОГО МОДЕЛИРОВАНИЯ SIMBIGRAPH

**Е.Б. Юдин, А.А. Курчанов (Омск)**

## 1. Введение

Система имитационного моделирования (СИМ) SIMBIGRAPH спроектирована для решения проблемы, поднятой в отчете европейского сообщества по развитию агентных вычислений [1] и заключающейся в необходимости интенсивного развития теоретических аспектов агентного моделирования, неотъемлемой частью которого является разработка новых моделей и методов структурной идентификации сетей взаимодействия агентов. Система поддерживает широкие возможности выбора структур взаимодействия агентов от 2D и 3D-решеток до случайных графов, возможность разработки имитационных моделей на языке программирования JAVA. Кроме того, система имеет мощную графоаналитическую поддержку за счет включения популярных библиотек JUNG, Colt и JFreeChart. Получить систему и примеры моделей можно, перейдя по адресу: https://github.com/yudinev/SIMBIGRAPH. Эти примеры разнообразны, среди них различные эпидемиологические модели, модель «Хищник-Жертва», модель «Сахарный мир», модель автокаталитической реакции Эрика Татары, модель для исследования сегрегации атомов на поверхности биметаллической частицы на подложке и в свободном состоянии [2] и другие.

## 2. Диаграмма классов СИМ SIMBIGRAPH

СИМ SIMBIGRAPH разработана в объектно-ориентированном стиле, на рис. 1 изображена диаграмма классов разработанной системы.

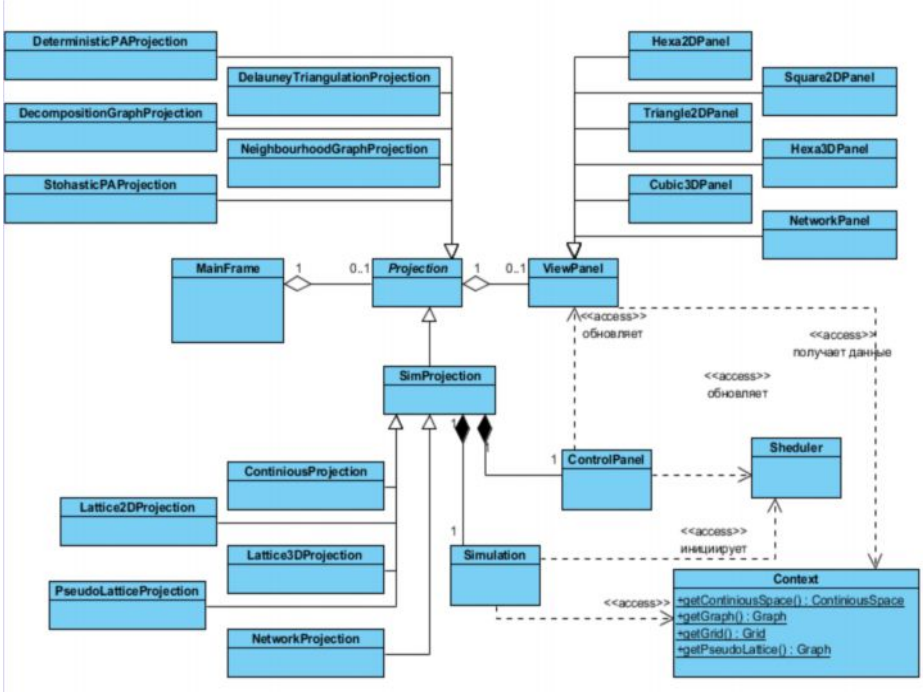


**Рис. 1. Диаграмма классов системы моделирования SIMBIGRAPH**

Абстрактный класс Simulation предназначен для упрощения создания моделей в СИМ SIMBIGRAPH. Любая пользовательская модель должна расширять класс Simulation. Только тогда написанный пользователем код будет автоматически скомпилирован при запуске модели. Класс Simulation предназначен для описания логики взаимодействия агентов и включает в себя основные методы ее визуализации. Для этих целей в классе Simulation поддерживаются следующие методы:

– void setSizeCell(int size) – устанавливает размер ячейки грида;

– Class[] getVisClass() – задает порядок отображения агентов (в решетке);

– Color getAgentColor(Object obj) – устанавливает цвет отображения агентов;

– abstract public void init(Object env) – инициализирует основную структуру моделирования (это может быть Grid или Graph). Инициализация обеспечивается до начала моделирования в методе Simulation.run();

– abstract Factory getAgentFactory() – задает способ создания агентов.

Для хранения параметров выбранной структуры взаимодействия агентов и непосредственно объекта модели предназначены классы, реализующие интерфейс Projection (Проекции), рис. 2. Структура и данные Проекций могут быть сохранены в файл формата XML и считаны из файла.

Реализованы два семейства Проекций.

1. Проекции, непосредственно реализующие интерфейс Projection. Для данного семейства Проекций не поддерживаются возможности агентного моделирования, нет доступа к классу Simulation. К Проекциям данного класса относятся: Decomposition graph – используется для генерации графов декомпозиции, Neigborhood graph – используется для генерации графов соседства, Delauney Triangulation – используется для построения триангуляции Делоне.

2. Проекции, наследуемые от класса SimProjection. К проекциям данного семейства относятся: 2D Lattices – плоская решетка, 3D Lattices – пространственная решетка, Networks – стандартные генераторы графов, Pseudo Lattices – пространственно распределенные структуры, Deterministic PA-graph, Stohastic PA-graph – графы предпочтительного связывания. Для данного семейства Проекций поддерживаются возможности агентного моделирования (доступ к классу Simulation), также при визуализации Проекции данного семейства отображается Панель управления (ControlPanel), рис. 2**2**.

### 3. Визуальный интерфейс СИМ SIMBIGRAPH

Класс MainFrame является классом главного окна программы (рис. 2) и содержит кнопку сохранения Проекции в файл и кнопку чтения из файла Проекций.

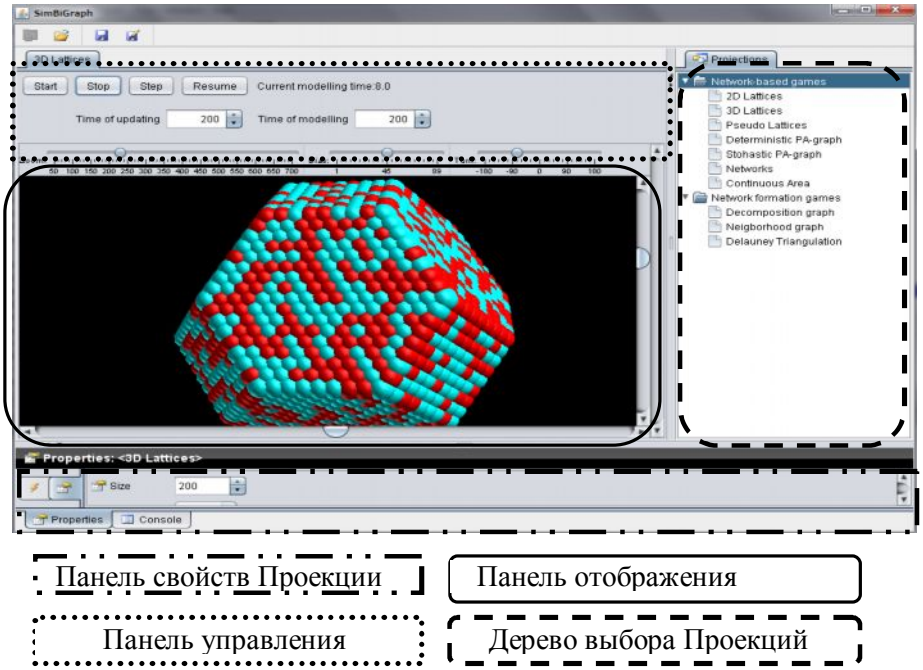


**Рис. 2. Внешнее представление основного окна программы SIMBIGRAPH**

Главное окно проекта, реализуемое классом MainFrame, содержит также:

– панель отображения, представленную классом ViewPanel (рис. 2), который используется для отображения моделируемой структуры. Класс ViewPanel содержит элементы управления свойствами внешнего вида. Класс ViewPanel наследуется классами Square2DPanel (отображается квадратная решетка), Hexa2DPanel (отображается гексагональная решетка), Triangle2DPanel (отображается треугольная решетка), NetworkPanel (отображается граф), непосредственно реализующими конкретную Панель отображения;

– панель управления (ControlPanel), которая реализует управление процессом моделирования: остановка, запуск, пошаговое выполнение модели;

– дерево выбора конкретной Проекции;

– панель установки свойств выбранной Проекции (выбор типа грида или случайного генератора графов, установка параметров генерации).

### 4. Последовательности действий при создании модели

Последовательность действий при запуске имитационной модели в SIMBIGRAPH представлена на рис. 3.

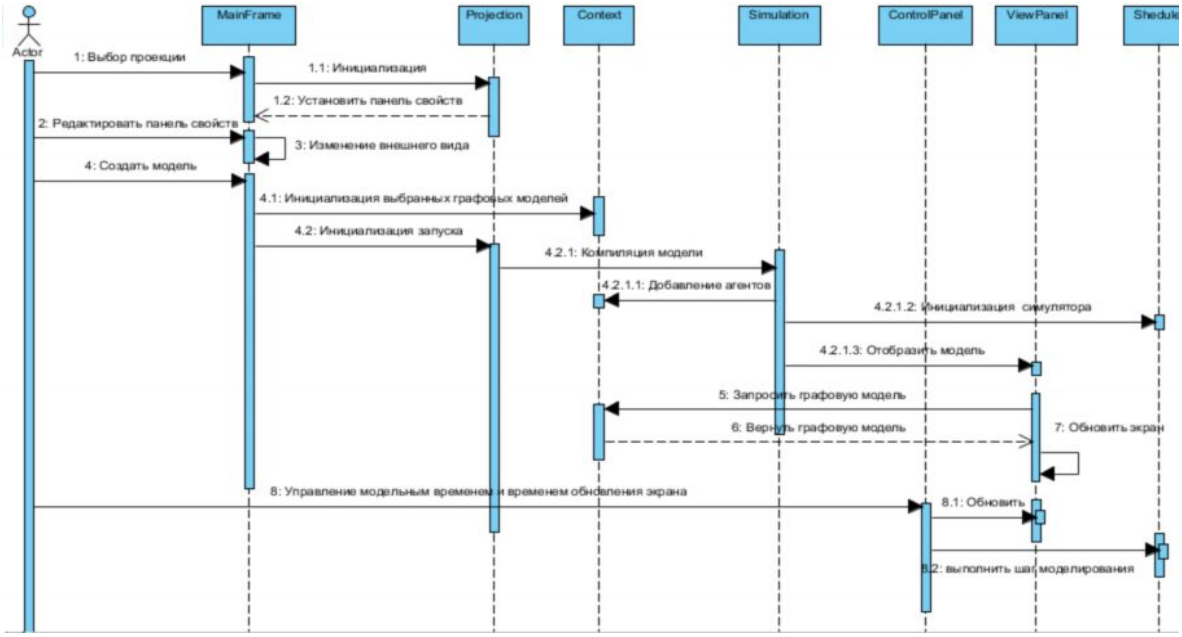


**Рис. 3. Диаграмма последовательности действий в SIMBIGRAPH**

Пользователь для начала работы должен выбрать одну из моделей, задающих структуру взаимодействия в дереве выбора Проекций. Выбранная Проекция включает свойства модели и параметры отображения (например, тип и размер решетки, графа и т.д.). При нажатии пользователем на кнопку создания модели происходит компиляция и инициализация модели. Управление свойствами Проекции производится с помощью панели управления и панели отображения.

Моделирование сетей в SIMBIGRAPH удобно описать с позиций двухэтапного подхода: на первом этапе формируется графовая модель, на втором – сформированная графовая модель используется для организации имитационного моделирования, т.е. моделирования передачи информации, ресурсов и т.д. На первом этапе осуществляется генерация случайных графов и задание размеров и типа грида (рис. 4). В СИМ SIMBIGRAPH реализованы генераторы графов предпочтительного связывания, графов Эрдеша-Реньи и т.д. Существует возможность генерировать графы с помощью триангуляции и генерации

графа соседства. На этом же этапе устанавливаются различные типы гридов: гексагональный, квадратный, треугольный, кубический. Вторым этапом является построение агентной модели, т.е. программирование агентов (параллельно функционирующих активных сущностей) и задание параметров симулятора (также называемого календарем событий или движком), устанавливающего причинно-следственные связи.

### 5. Пример разработки модели

Рассмотренный подход построения агентной модели абстрагируется от топологии взаимодействия агентов. Соседство (задается на основе смежности ячеек) устанавливается посредством реализованных специальных классов, которые эффективно и однозначно определяют множество соседних агентов. Например, для модели, представленный на рис. 4 (плоский XOR автомат), класс клеточного автомата – CA.java программируется с использованием дополнительного класса для обращения к соседним ячейкам грида – MooreQuery.java.

```
class CA {
          public int state, oldState;
          public void switchStates(){oldState = state;}
          public void step(){
                    Grid grid = Context.getGrid();
                    MooreQuery<CA> query = new MooreQuery(grid,this);
                    Iterator<CA> iter = query.query().iterator(); int a = iter.next().oldState; state=a;
                    while (iter.hasNext()) {a=state; state=(a^iter.next().oldState);}
          }
}
```

**Рис. 4. Реализация класса CA.java для моделирования клеточного XOR автомата**

В листинге кода модели XOR автомата (рис. 5) устанавливается способ отображения агентов и синхронизируется их одновременное выполнение (рис. 6).

```
public class SimulationCA extends Simulation {
          class CA {// задание начального состояния
          private void doStructure() {
                              Grid grid = Context.getGrid();
                              GridDimensions dim = grid.getDimensions();
                              for (int i = 0; i < dim.getWidth(); i++)
                                        for (int j = 0; j < dim.getHeight(); j++) {
                                                  CA ca = new CA();    grid.moveTo(ca, i, j);}
                              int seedWidth = dim.getHeight() / 5;
                              for (int j = dim.getHeight() / 2 - seedWidth / 2; j < dim
                                                  .getHeight() / 2 + seedWidth / 2; j++) {
                                        for (int i = dim.getWidth() / 2 - seedWidth / 2; i < dim
                                                            .getWidth() / 2 + seedWidth / 2; i++) {
                                                  CA ca = (CA) grid.getObjectAt(i, j);
                                                  ca.setState(1);
                                        }
                              }
          return;}
          public void step(SimState state) {
                              Grid grid = Context.getGrid();
                              GridDimensions dim = grid.getDimensions();
                              for (int i = 0; i < dim.getDimension(0); i++) {
                                        for (int j = 0; j < dim.getDimension(1); j++) {
                                                  CA ca = (CA) grid.getObjectAt(i, j);
                                                  ca.setOldState();}
                              }
                              for (int i = 0; i < dim.getDimension(0); i++) {
                                        for (int j = 0; j < dim.getDimension(1); j++) {
                                                  CA ca = (CA) grid.getObjectAt(i, j);
                                                  ca.step(); }
                              }
          }
          public void init(Object env) {  doStructure();
          } // метод инициализации модели
          public void start() {
                    super.start();
                    schedule.scheduleRepeating(this);
          }
          public Color getAgentColor(Object obj) {// определение цвета ячейки
                    Color col = Color.GREEN;
                    if (obj instanceof CA)
                              if ((((CA) obj).state == 1)
                                        return Color.BLACK;   else return Color.WHITE;
          return col;}
          @Override public Factory getAgentFactory() {
return null;}}
}
```

**Рис. 5. Реализация класса SimulationCA модели XOR-автомата в SIMBIGRAPH**

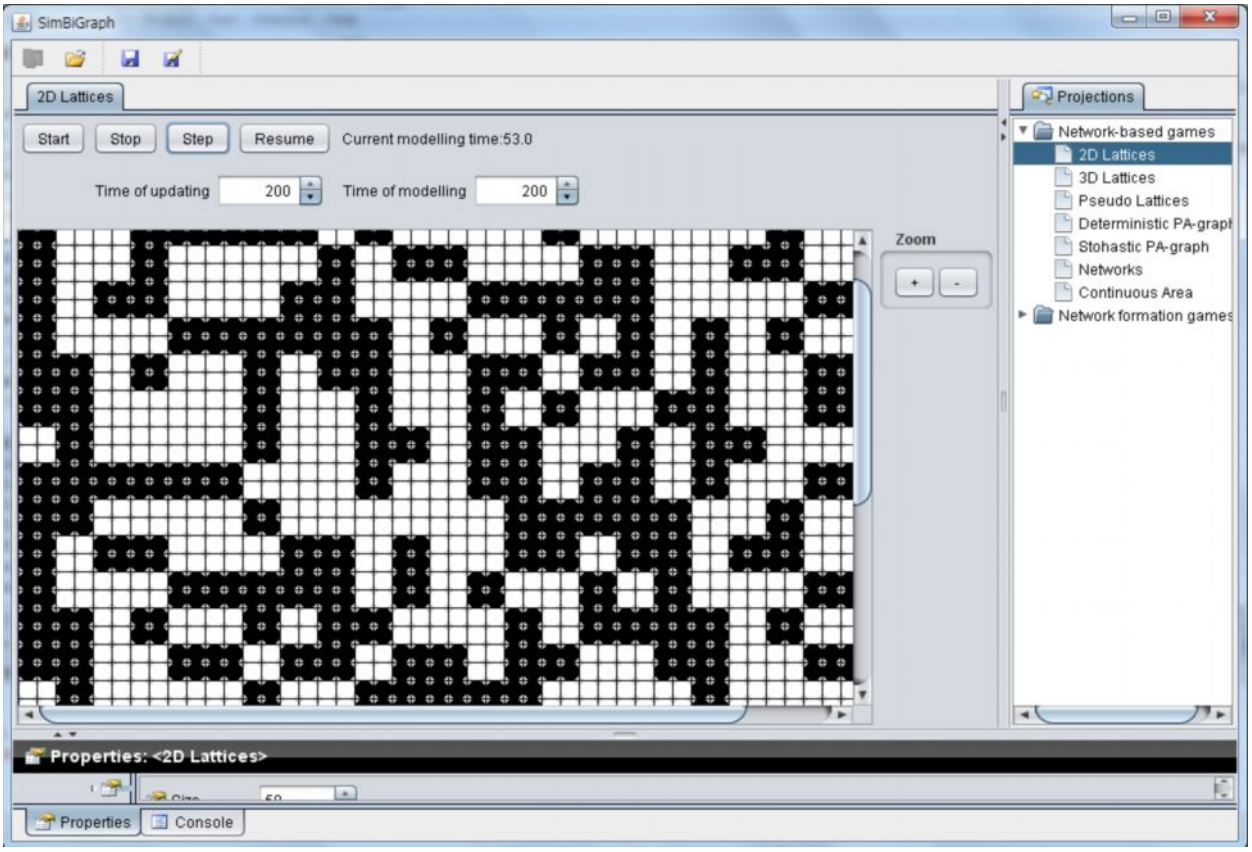


**Рис. 6. Визуализации модели клеточного XOR автомата**

### 6. Заключение

СИМ SIMBIGRAPH предназначена для решения актуальных задач системного анализа больших сетей и решеток. Разработка СИМ SIMBIGRAPH позволила пользователям использовать последние достижения авторов системы в области имитационного моделирования и теории случайных графов. В работах [2-5] SIMBIGRAPH используется в качестве базового инструмента анализа сложных процессов в сетях и решетках. На конференции ИММОД-2011 А. Ю. Гарбаревым была представлена веб-система BridgeGL [6], поддерживающая модели, разработанные в SIMBIGRAPH. Как и СИМ широкого назначения Simulab [7], СИМ Simbigraph является примером удачной разработки в лаборатории ИМСАИТ ОмГТУ.

### Литература

1. Roadmap of AgentLink III, the European Coordination Action for Agent-Based Computing (IST-FP6-002006CA) edited by Michael Luck et al., AgentLink III, 2005. – 108 p.
2. **Задорожный В.Н., Юдин Е.Б.** Мультиагентное моделирование микромира // Информационные технологии и автоматизация управления: матер. межвуз. науч.-практ. конф. – Омск: Изд-во ОмГТУ, 2009. – С. 186–188.
3. **Ганеева М.И., Огнев Д.А., Пендер Е.А.** Построение агентных моделей в системе моделирования SIMBIGRAPH // Информационные технологии и автоматизация управления, матер. регион. конф. – Омск. – 2011. – С. 103–105.
4. **Коробов В. В.** Создание среды для последующего моделирования вирусного маркетинга // Прикладная математика и фундаментальная информатика; сб. науч. трудов. – Омск, 2011.– С. 103–105.

5. **Юдин Е.Б., Задорожный В.Н** Агентная модель «хищник-жертва» на статистически однородных структурах // Информационные технологии и автоматизация управления: матер. межвуз. науч.-практ. конф. – Омск: Изд-во ОмГТУ, 2009. – С. 205–207.
6. **Гарбарев А.Ю.** Реализация системы агентного имитационного моделирования в WEB // Имитационное моделирование. Теория и практика: материалы 5-й всероссийской конференции. Том 2. – СПб: ФГУП ЦНИИТС. – С. 346–349.
7. **Ершов Е.С., Юдин Е.Б.** Система имитационного моделирования Simulab / Имитационное моделирование. Теория и практика: матер. 4-й всеросс. конф. ИММОД-2009. – СПб., 21–23 октября 2009 г. – СПб. – 2009. – Т. 1. – С. 257–261.