ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

УДК 004.75

Д.В. Жевнерчук, Д.А. Лопатин

ПРИМЕНЕНИЕ АГЕНТНОГО ПОДХОДА К ПРОЕКТИРОВАНИЮ И РЕАЛИЗАЦИИ МОДЕЛЕЙ САМООРГАНИЗАЦИИ КОНФИГУРАЦИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Нижегородский государственный технический университет им. Р.Е. Алексеева

Приводится обоснование выбора агентного подхода в качестве базового для моделирования и исследования процессов балансировки нагрузки в вычислительных сетях. Предложена архитектура модели вычислительной сети, основанная на агентном подходе с применением интеллектуальных агентов.

Ключевые слова: агентный подход, интеллектуальный агент, вычислительная сеть, событийная модель, запрос, диаграмма деятельности.

Пусть *S* – распределенная вычислительная система, предоставляющая некоторое множество сервисов, меняющееся с течением времени, и обладающая высоким уровнем интероперабельности с внешними системами, являющимися потребителями сервисов. Такая система в процессе функционирования должна реорганизовывать и адаптировать свои ресурсы под меняющуюся нагрузку, вызванную клиентскими запросами, появлением новых сервисов, выходом из строя вычислительных узлов и др.

Известен ряд работ, в которых предлагаются модели процесса распределения нагрузки на основе сетей СМО, коллективов вычислителей и другого, а также алгоритмы балансировки нагрузки [1,2]. Основной чертой полученных результатов является ориентация на частные случаи вычислительных систем, такие как кластеры и GRID, и для применения их в *S* требуются дальнейшие исследования.

Постановка задачи

Для проведения эффективных исследований процессов балансировки нагрузки *S* необходимо разработать подход к проектированию и реализации среды моделирования *S*, обосновать выбор используемых базовых подходов. Решение должно содержать модели проектирования S, отражающие ее различные структурные и функциональные аспекты. Объект исследования может быть отнесен к классу сложных самоорганизующихся адаптивных систем [3], который был введен и изучен Дж. Холландом, поскольку в разных формах существуют механизмы агрегации групп элементов, нелинейности, модификации ресурсов и информации, модификации структуры и поведения ее элементов, влияющих на эмерджентные свойства *S*, что приводит к повышению ее устойчивости. Известно, что естественным подходом для описания систем этого класса является агентный подход.

Таким образом, требуется адаптировать агентный подход к построению моделей распределенных вычислительных сервис-ориентированных систем с меняющимся составом сервисов и высокими показателями интероперабельности с клиентскими системами.

Методика

Далее описаны наиболее значимые архитектурные решения, использованные при создании модели *S*. Методической основой решения выступают объектно-ориентированный и агентные подходы.

Для моделирования структуры вычислительной системы применяется граф, в котором вершины – это узлы, а ребра – сеть, соединяющая узлы. Генерация графа вычислительной сети основана на шаблоне «Абстрактная фабрика» (англ. Abstract factory) – порождающем шаблоне проектирования, позволяющем изменять поведение системы, варьируя создаваемыми объектами, сохраняя при этом интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов [4].

Применены следующие абстрактные классы: AbstractNode и AbstractEdge – абстрактные классы, декларирующие методы, реализация которых будет определять поведение конкретных Node и Edge, AbstractNetworkFactory – абстрактный класс, имплементация которого (конкретная фабрика) определяет, из каких узлов состоит сеть и связи между ними. Для фабрики разработан объект Client, который использует ее для генерации вычислительной сети. Взаимодействует клиент с фабрикой и созданными ею объектами через интерфейсы, оперируя абстракцией, независимо от реализации. Абстрактная фабрика создания модели вычислительной сети описана диаграммой классов, представленной на рис. 1.

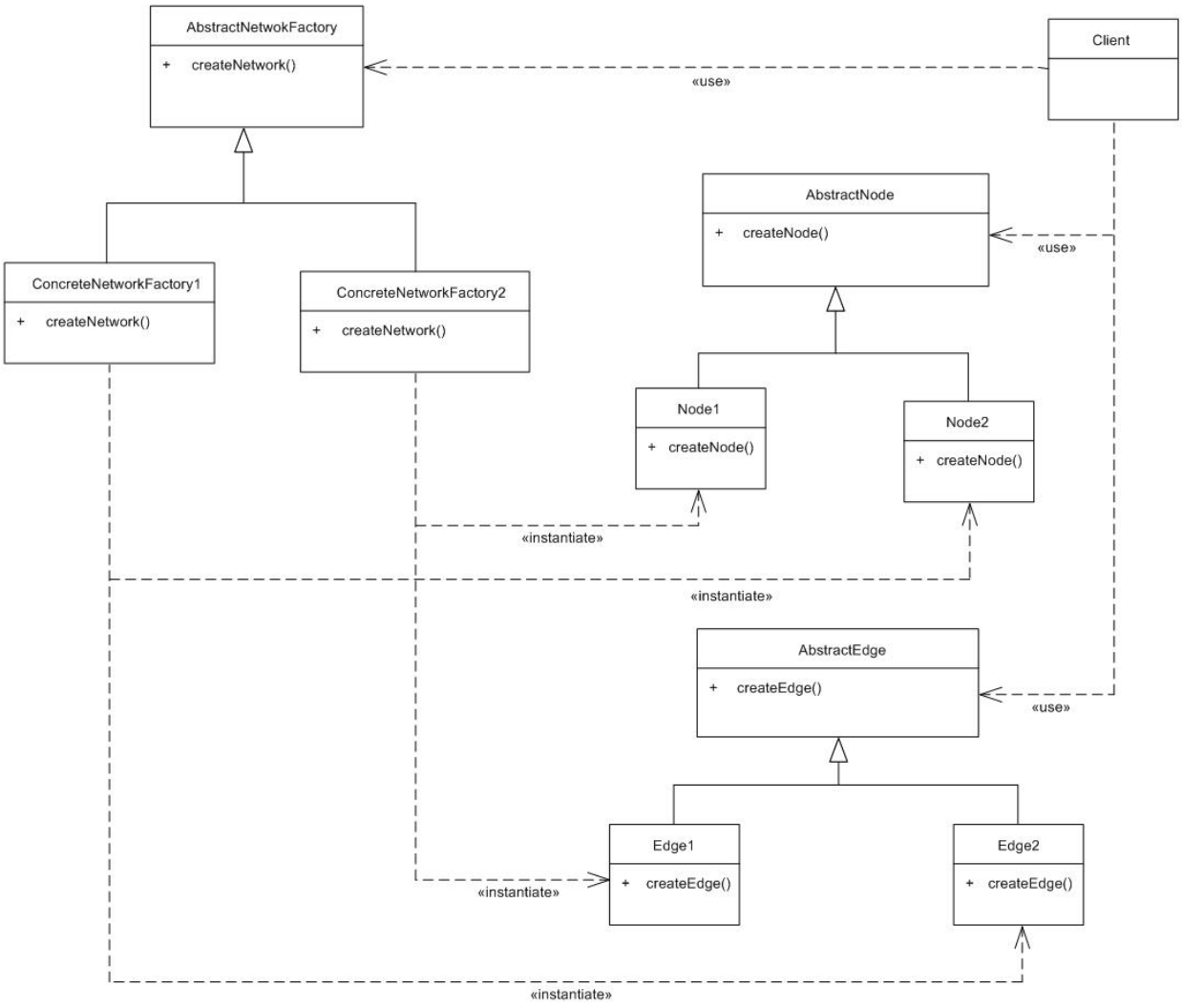

Рис. 1. Абстрактная фабрика создания модели вычислительной сети

Благодаря данному шаблону мы можем получить неоднородную сеть, в которой узлы могут обладать различным поведением, также появляется возможность настраивать взаимодействие между узлами и моделировать различную нагрузку на сеть. Одним из плюсов является возможность расширения предложенной реализации посредством добавления объектов с новым поведением.

Агенты расположены на узлах и представляют собой самостоятельные сущности, представленные в виде потоков. Они осуществляют переодический опрос (polling) узла, с целью поддержания информации о доступных ресурсах в актуальном состоянии. В модели количество агентов определено сгенерированной вычислительной сетью, поддерживается механизм межагентного взаимодействия. Каждый узел предоставляет информацию о доступных агентах, которыеза ним закреплены.

Для создания агентов предложено использовать пораждающий шаблон проектирования «Фабричный метод» (англ. Factory Method). Данный паттерн обладает базовыми функциями абстрактной фабрики, но в отличии от последней формирует лишь один интересующий объект, а не группу связанных. Фабрика делегирует создание объектов наследникам родительского класса, что позволяет манипулировать абстрактными объектами на более высоком уровне.

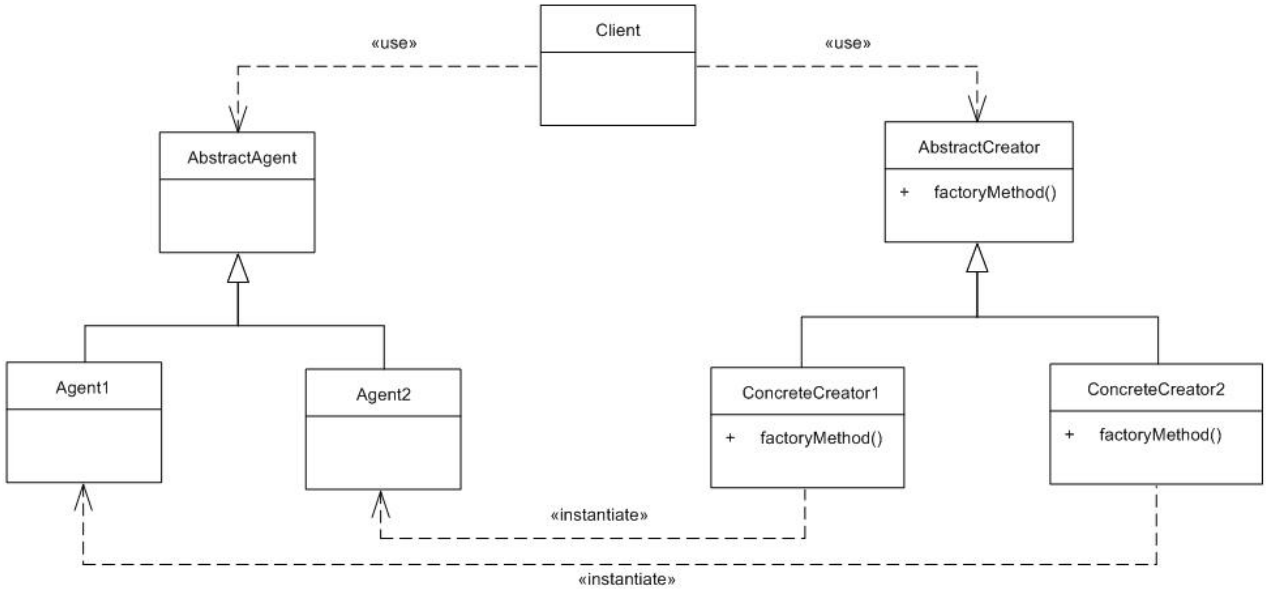

Рис. 2. Фабричный метод создания агентов с различным поведением

Для получения входных данных (конфигурации модели) и последующего их хранения предложено использовать шаблон проектирования DAO (data access object или объект доступа к данным) — это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Определённые возможности предоставляются независимо от того, какой механизм хранения используется, без необходимости специальным образом соответствовать этому механизму хранения [5]. ModelConfigurationDAO – это общий интерфейс, декларирующий методы, которые описывают операции, применимые к конфигурации модели. В нашем случае достаточно стандартного набора операций – CRUD (create, read, update, delete).

Объект доступа к данным (рис. 3) использует класс Connection, который является объектом, описывающим доступ к определенному механизму хранения.

Описание создания и хранения кофигураций представляется в виде XML файлов (XMLModelConfigurationDAO). Разработана xsd-схема, содержащая набор правил, по которым строится XML-файл. При реализации такого подхода использовались стандартные

средства языка JAVA, которые предоставляют удобный инструмент для серелизации/десерелизации объектов на основе xsd. Также данный подход позволяет проводить валидацию конфигурации, основываясь на правилах, описанных в xsd-схеме.

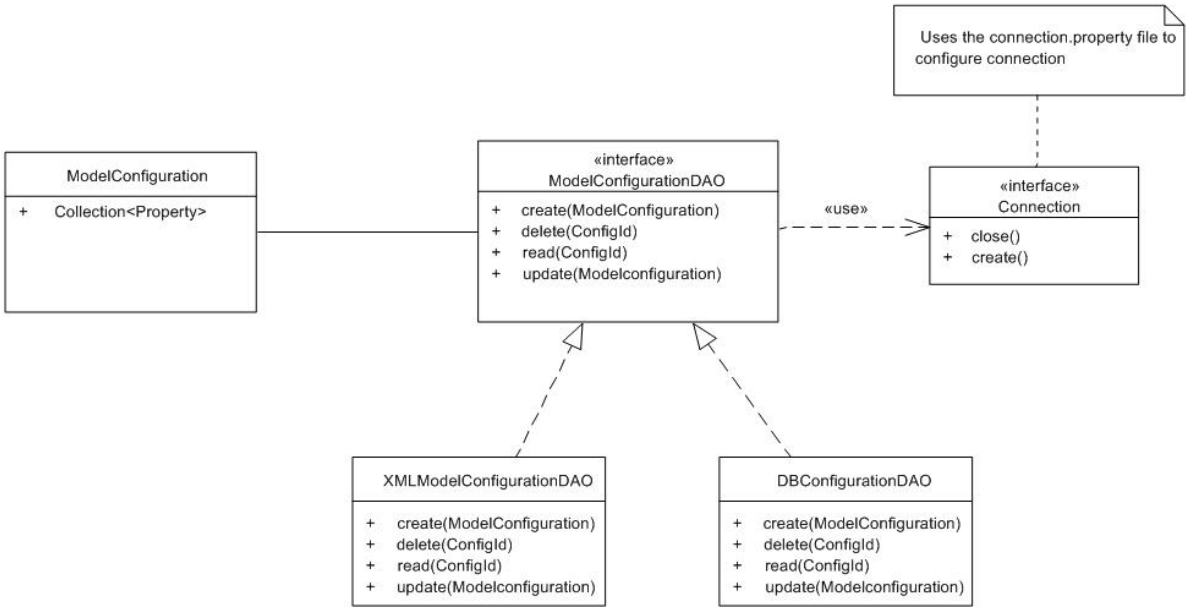

Рис. 3. Объект доступа к данным конфигурации модели

Для отслеживания состояния модели во время эксперимента применяется журнал (log) событий. Основываясь на том, что логгирование будет происходить в многопоточной среде, предлагается использовать паттерн Singleton (одиночка) со статической инициализацией, что гарантирует его потокобезопасность [6] (рис. 4).

Использование данного шаблона подразумевает, что при старте системы будет создан всего лишь один экземпляр данного класса и каждая сущность системы будет иметь ссылку на него. Логгирование может производиться на определенном уровне (в зависимости от детализации):

1. INFO – логгирование основных событий системы.
2. DEBBUG – логгирование всех событий системы.
3. ERROR – логгирование ошибок.

Настройка уровеня логгирования также относится к конфигурации модели.

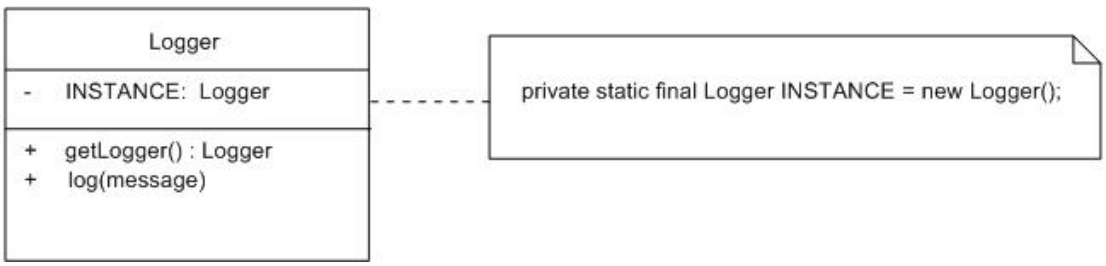

Рис. 4. Логгирование событий системы(шаблон одиночка)

Важной частью системы является сбор статистики. Используется подход к организации сбора значимых показателей модели в форме, пригодной для дальнейшего анализа или даже облегчающей его (рис. 5). Основываясь на поставленных задачах, предложено осуществлять сбор необходимых показателей в виде CSV-файлов (англ. Comma-Separated Values – значения, разделённые запятыми) – текстовый формат, предназначенный

для представления табличных данных. Каждая строка файла – это одна строка таблицы. Значения отдельных колонок разделяются разделительным символом, обычно запятой. Существует множество инструменов, позволяющих анализировать и визуализировать данные, которые хранятся в таком формате.

Абстрактный класс DataCollector предоставляет возможность реализующим его классам выполнять задачу сбора статистики по расписанию в отдельном потоке. Конкретные реализации этого абстрактного класса осуществляют сбор статистики для определенной группы объектов. Например, AgentDataCollector собирает статистику по всем агентам системы.

Любая сущность (узел, клиент, агент) каждый момент модельного времени сохраняет состояние своих ключевых показателей внутри себя (поскольку записывать в файл каждую единицу модельного времени неэффективно). Каждый DataCollector знает о соответствующих ему сущностях и, в соответствии с настроенным интервалом, последовательно осуществляет опрос (poll) всех сущностей и записывает полученные данные в определенный CSV-файл. После опроса сущности очищают сохраненные показатели, освобождая память.

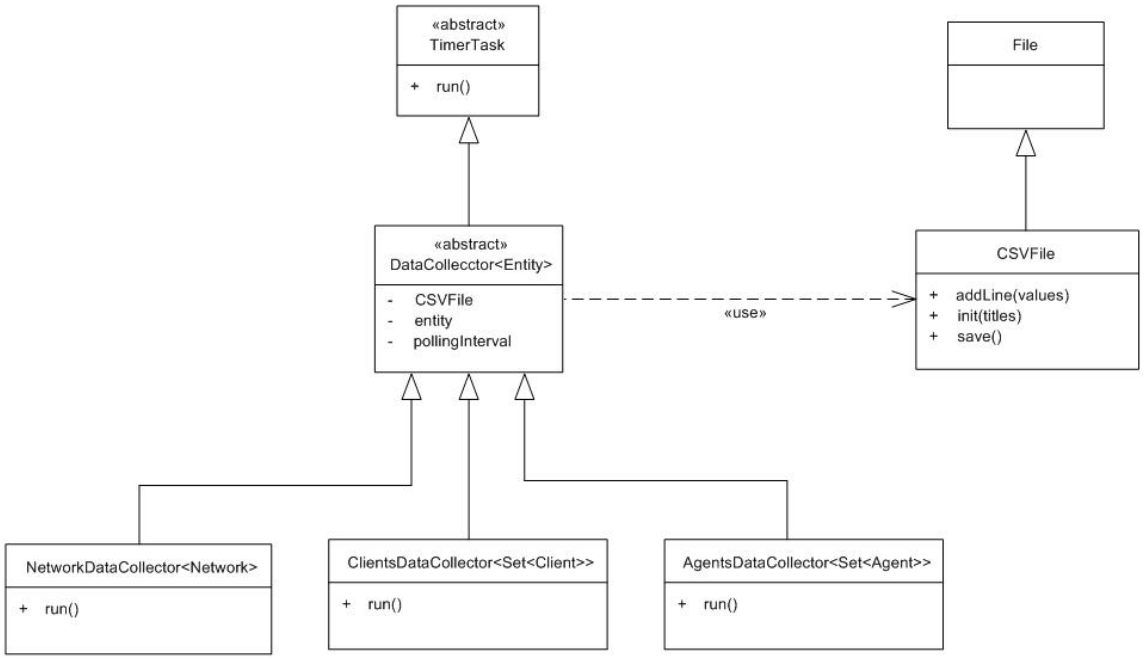

Рис. 5. Сбор ключевых показателей системы

Клиент – это сущность системы, которая с определенной периодичностью, в зависимости от кофигурации модели, отправляет запросы на сервер. Поведение клиента основывается на событийной модели (рис. 6).

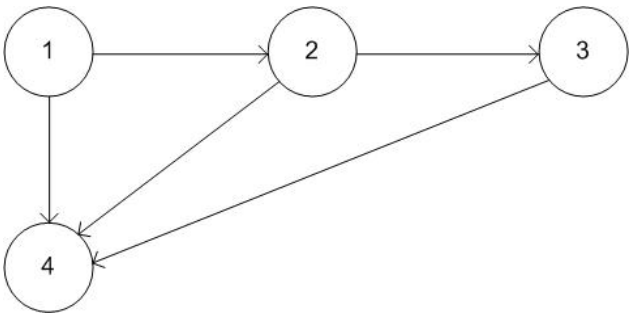

Рис. 6. Сбор ключевых показателей системы

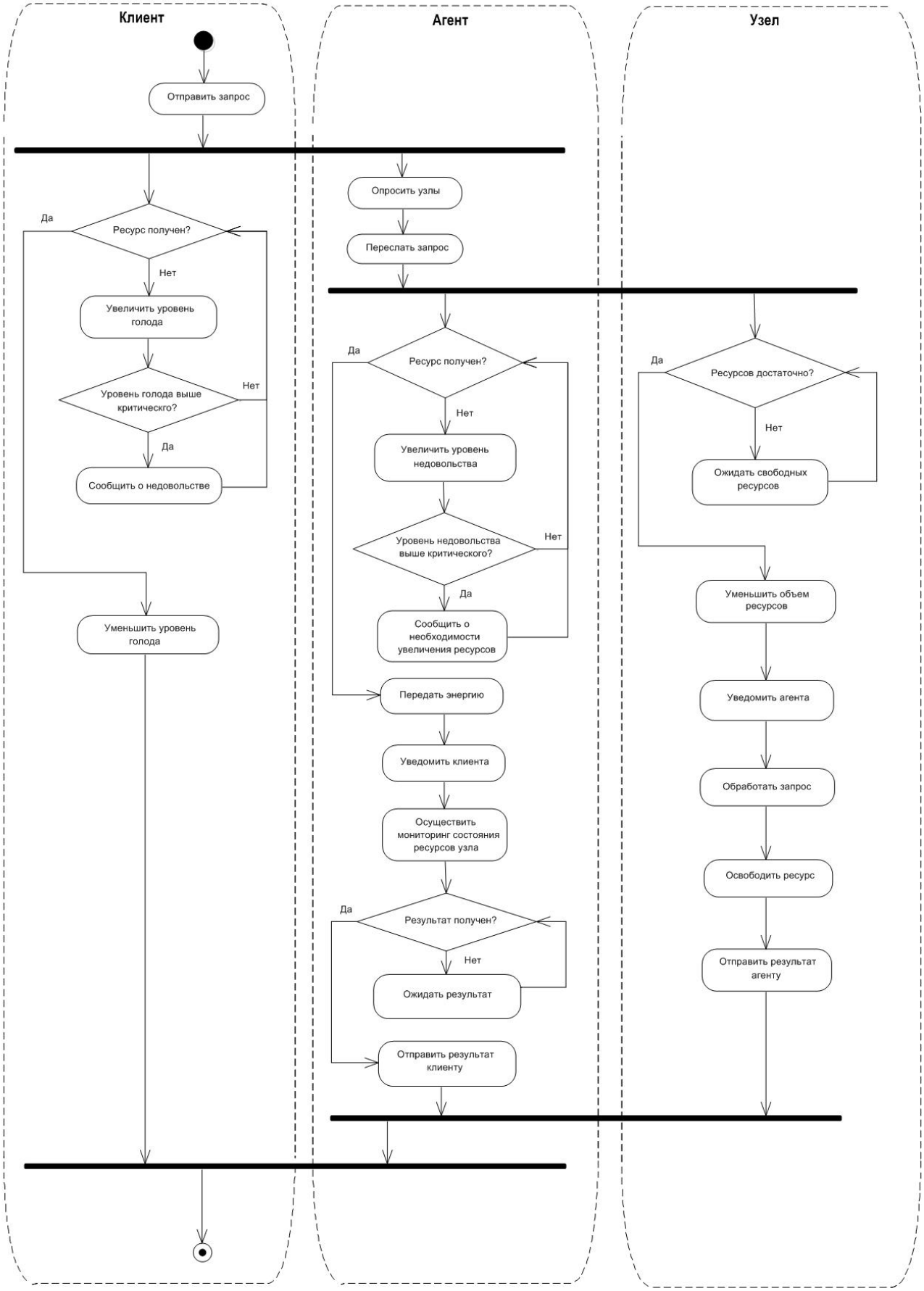

Рис. 7. Диаграмма деятельности системы

Всего выделено четыре события, при наступлении которых клиент выполняет определенные действия:

1. Отправка запроса на сервер. Клиент отправляет запрос на сервер, указывая количество необходимых ему ресурсов и интервал времени, на который они ему необходимы. После отправки запроса клиент начинает увеличивать чувство голода до тех пор, пока ресурс не будет предоставлен в полном объеме.

2. Уровень голода превышает критический уровень. С момента отправки запроса уровень голода клиента повышается и может превысить допустимый уровень, в этом случае клиент «жалуется» на обслуживание, т.е. сообщает менеджеру ресурсов о своем недовольстве. Менеджер ресурсов, в зависимости от числа неудовлетворенных клиентов, принимает решение о добавлении нового узла в сеть.

3. Создан новый узел. Под «давлением» неудовлетворенных клиетов менеджер принимает решение увеличить предоставляемый ресурс, добавив новый узел в сеть, о чём и оповещает голодных клиентов. Клиенты, в свою очередь, выгружают заявку с узла, который так и не смог предоставить необходимый ресурс в полном объеме, и отправляют запрос на только что сформировавшийся узел.

Получен ответ, предоставлен ресурс. Агент информирует пользователя, что узел может полностью удовлетворить его требования. Клиент проверяет, была ли реализована заявка вовремя. Если да, то он уменьшает количество накопленного голода с момента отправки заявки.

Разработана диаграмма деятельности (рис. 7). Клиент посылает запрос в сеть. Узел, который будет обрабатывать данный запрос, первоначально выбирается случайным образом или зависит от географического положения (выбирается ближайший к клиенту узел). После отправки запроса, ожидая получения необходимых ресурсов, узел начинает испытывать голод, который увеличивается с каждой единицей модельного времени.

На каждом узле закреплен интеллектуальный агент, который следит за его состоянием и производит мониторинг состояния ресурсов данного узла. Также агент осведомлен о других, соседних агентах, с которыми может контактировать. Агент опрашивает связанный с ним и соседние узлы для получения сведений о текущем состоянии феромона. После оценки полученной информации агент принимает решение о привлекательности узлов (чем феромон выше – тем выше привлекательность), также на уровень феромона оказывает влияние загруженность сети. Агент пересылает запрос, либо обрабатывает его сам.

Основываясь на информации о состоянии ресурсов узла, агент отправляет заявку на обработку, либо добавляет в очередьдо тех, пор пока ресурс не сможет в полном объеме удовлетворить заявку. После того, как заявка попадает на обработку, узел уменьшает объем предоставляемых им ресурсов, и оповещает об этом своего агента. В свою очередь, агент передаёт узлу часть своей энергии и оповещает клиента о том, что ресурс предоставлен. Далее агент производит мониторинг состояния ресурсов узла для поддержания информации об уровне выделяемого феромона в актуальном состоянии.

При увеличении уровень голода клиента может превысить критическую отметку, при этом клиент информирует менеджера ресурсов о неудовлетворительном обслуживании. Менеджер ресурсов на основании жалоб клиентов может увеличить количество ресурсов, добавив новый узел в сеть. После обработки заявки узел освобождает ресурсы и уведомляет агента о результате обработки. Агент пересылает результат клиенту.

Предложенная реализация событийной модели и диаграммы деятельности системы базируется на методике самоконфигурирования пространственно-временного ресурса, которая была разработана на кафедре ВСТ ИРИТ НГТУ им. Р.Е. Алексеева.

Экспериментальная часть

Для исследования модели была разработана мультиагентная система с применением Java-технологий в среде Eclipse.

Проведена проверка свойств открытости предложенной архитектуры. Свойство переносимости моделей обеспечено механизмами JVM. Построенная модель функционирует на широко распространенных платформах: win, *nix. Проверена расширяемость платформы по:

- новым видам собираемой статистики,
- количеству агентов, принадлежащих отдельным группам,
- новым типам узлов и клиентов, имитируемых агентами и др.

Обеспечена интероперабельность с системами управления базами данных: MS SQL Server 2005 и выше, MySQL 5, Postgreesql. Тестирование совместимости с другими СУБД не проводилось. Обеспечена интероперабельность со средствами анализа данных: Weka, RapidMiner Studio, MS Analysis Services.

В ходе серии экспериментов удалось добиться сходимости к конечному значению количества источников ресурсов. При этом вариация ресурса в стационарном состоянии составила в среднем от 7% до 19% от разброса между максимальным и минимальным количеством ресурса, наблюдаемым за весь период моделирования.

При реализации данной модели активно использовались как общие принципы проектированния GRASP (англ. General Responsibility Assignment Software Patterns), так и более конкретные шаблоны GOF, благодаря которым архитектура системы является прозрачной и понятной. Использование шаблонов даёт нам возможность легко изменять систему и масштабировать её. Архитектурные решения, описанные в этой статье, предоставляют возможность повторного использования кода.

Библиографический список

1. **Хританков, А.С.** Модели и алгоритмы распределения нагрузки. Алгоритмы на основе сетей СМО // Информационные технологии и вычислительные системы. 3/2009. С. 33-48.
2. **Хританков, А.С.** Модели и алгоритмы распределения нагрузки. Модель коллектива вычислителей. Модели с соперником, Информационные технологии и вычислительные системы. 2/2009. С. 65–80.
3. **Holland, J.H.** Hidden Order: How Adaptation Builds Complexity. NewYork: HelixBooks (AddisonWesley), 1995.
4. Erich Gamma, Richard Helm, Ralph Johnson, JohnVlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.
5. **Bloch, Joshua.** Effective Java Second Edition / Joshua Bloch. Prentice Hall, 2008.
6. **Goetz, Brian,** Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea / Brian Goetz [et al.] // Java Concurrency in Practice. Addison-Wesley Professiona, 2006.

Дата поступления
в редакцию 27.06.2014

D.V. Zhevnerchuk, D. A. Lopatin

APPLICATION OF AGENT-BASED APPROACH TO DESIGN AND IMPLEMENTATION OF THE COMPUTER SYSTEM CONFIGURATION SELF-ORGANIZATION MODELS

Nizhny Novgorod state technical university n.a. R.E. Alexeev

Subject: The subject of this study is the agent-base approach using in open computer systems model architecture.
Purpose: The aim is to create architecture and design templates for open computer systems modeling and researching.
Design/methodology/approach: A theoretical framework is proposed based on software design patterns, methodology of open systems in computer science and UML.
Findings: The results can be applied to the design and researching of scalable, extensible integrated information systems whose components operate on different hardware and software platforms.
Research limitations/implications: The present study provides a continuation and development simulation of self-organization processes in distributed spatiotemporal resources of open computer systems researching.
Originality/value: The paper describes an approach to open computing systems' properties modeling and study with using biological systems' self-organization mechanisms. It could be used in researching of open computer systems and in load balancing systems design.

Key words: agent-based approach, an intelligent agent, computer network, event model, query, activity diagram.