Математические структуры и моделирование
2010, вып. 21, с. 44–54

УДК 004.4'2

ВЫСОКОУРОВНЕВАЯ МОДЕЛЬ СЕМЕЙСТВА ПРОГРАММНЫХ КОМПОНЕНТОВ ДЛЯ ПОДДЕРЖКИ ЗАНЯТИЙ В ИГРОВОЙ ФОРМЕ

С.В. Гусс

В работе представлена классификация элементов повторного использования, указано место в ней каркаса как основы для разработки семейства программных продуктов. Даётся краткое описание его отличия от других представителей классификации. Приводится описание высокоуровневой модели разработанного автором каркаса в виде диаграмм языка UML и текстовых пояснений.

Введение

С тех пор как компьютерные программы стали рассматриваться разработчиками как инженерные творения и создавались уже не единичные программы, решающие конкретную проблему, а продукты, принадлежащие определённому программному семейству, проблема повторного использования приобрела большое значение. О возможности создания семейства программных продуктов, в отличие от единичных программ, говорил в одной из своих статей Давид Парнас [1].

На протяжении многих лет занятия в игровой форме, например лингвистические, давали результат в обучении не только языку и его особенностям, но и применению его в профессиональной сфере. Об этих занятиях можно говорить так же, как и о других обучающих занятиях в игровой форме. Большая работа в этом направлении проделана Марком Пренским, исследователем в области электронного обучения, в основе которого лежат компьютерные игры. В одной из его публикаций [2] можно найти информацию о том, почему именно игровая форма является привлекательной. Ниже представлен список причин, взятый из книги.

1. Игры доставляют удовольствие.
2. Способствуют интенсивному вовлечению в процесс обучения.
3. В них заложены правила, из чего следует упорядоченность и структура процесса.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: InfoGuss@gmail.com

4. Они мотивируют, так как имеют определённые цели.
5. Они интерактивны, что заставляет действовать.
6. У них обязательно есть обратная связь, что способствует обучению.
7. Для игрового процесса характерно свойство привыкания, что позволяет без труда получать новые знания посредством игры.
8. Победа, одержанная в игре, даёт возможность почувствовать свои силы.
9. Они развивают творческие способности благодаря тому, что заставляют справляться с проблемами, искать способы их решения.
10. Игры способны развить коммуникационные способности благодаря специальным многопользовательским режимам.
11. Могут воздействовать на эмоции.

В этой же публикации есть описание возможных игровых стилей для различных тематических направлений изучаемых дисциплин.

1. Место каркаса в классификации элементов повторного использования

Часто в литературе вместо фразы «элементы повторного использования» можно встретить словосочетание «компоненты многократного применения». Они не являются синонимами, но это описание одного и того же явления на разных уровнях представления. В первом случае речь идёт об абстрактном уровне представления, в последнем - о конкретном, или уровне реализации, когда имеют в виду наличие готового к использованию компонента, а не описывающую его модель. Элементы повторного использования можно классифицировать по нескольким признакам.

1. По уровню представления:
   - Модели. Представляют собою высокоуровневое описание структуры и поведения.
   - Набор кода. Готовый к применению код на определённом языке программирования.
   - Исполняемые модули. Компоненты, готовые к запуску пользователем или другим программным приложением. Характеризуются отсутствием привязки к языку программирования. Но возможна привязка к определённой программно-аппаратной платформе.
2. По специфике повторно используемого элемента. Классификация похожа на ту, что приведена в [3]. Только в представленной ниже классификации не фигурирует такой элемент, как модель, т.к. она не подходит под данный признак деления, в связи с чем этот элемент был отнесён к другой группе.
   - Функциональные библиотеки (набор подпрограмм под специфические нужды). В отличие от каркасов здесь повторно используются функции, а написание тела приложения, в котором будут использованы эти функции, остаётся задачей разработчика.
   - Образцы (соответствуют этапам разработки):
     – анализа [4] (применимы для анализа предметной области прило-

жения);
  - архитектурные [5] (для упорядочивания архитектуры и реализации определённых характеристик качества);
  - проектирования [6] (для применения опыта, накопленного и обобщённого другими разработчиками в виде особых проектных решений);
  - реализации [7] (для обслуживания задач ежедневного программирования, а также для придания программному коду читабельного вида и внесения в него большей ясности).
- Каркасы. Более специализированы, чем образцы. Важно отметить, что приложение может разрабатываться на основе нескольких каркасов; это довольно часто относится к крупным приложениям масштаба предприятия. Особенность состоит в том, что повторно используется тело приложения, к которому добавляются функции и уточнения.

3. По этапу разработки (элементы могут быть представлены и моделями, и набором кода, и в виде исполняемых модулей, и даже просто в виде текстового описания):
   - Предметной области.
   - Сбора и анализа требований.
   - Архитектуры. Чтобы архитектура могла быть многократно используемой, крайне необходимо отделить её от реализации, а на основании [8] - ещё и от алгоритмов и представления данных.
   - Проекта.
   - Реализации.
   - Тестирования.

2. Проблема разработки каркаса

Каркас – это «набор взаимодействующих классов, составляющих повторно используемый дизайн для конкретного класса программ» [6]. Конкретный класс программ, или семейство программных продуктов – набор программ, имеющих так много общего, что выгоднее изучить их общие аспекты, прежде чем изучать те аспекты, в которых они различны [1]. Влияние каркаса на детализирующую его систему может быть описано следующими принуждающими параметрами [6]:

1. Определённая архитектура.
2. Структура классов и объектов.
3. Основная функциональность, описанная в классах.
4. Методы взаимодействия между объектами.
5. Потоки управления.

Необходимость этих параметров обусловлена тем, что разработчики могут сконцентрироваться на специфике разрабатываемой программы и не тратить время на поиск черт, уже описанных и известных. Одна из проблем дизайна любого типа программного обеспечения заключается в том, что, как отмечается в [6], дизайн должен соответствовать задаче, но в то же время быть общим для того,

чтобы была возможность учесть требования, возникновение которых возможно в будущем. Сложность состоит ещё и в том, чтобы разложить систему на объекты. Кроме того, «вся система должна обладать концептуальным единством» [9].

3. Описание модели каркаса

Концепция. Главными пользователями каркаса являются разработчики программного обеспечения. Реализация модели каркаса, описываемой в данной работе, позволяет ускорить процесс разработки компонентов программного продукта за счёт того, что в нём уже заложена стратегия управления всеми необходимыми операциями, действия которых происходят в игровом процессе. Разработчику, использующему каркас, необходимо лишь уточнить в нём некоторые детали и заложить отдельные стратегии реализации операций, специфичных для решаемой задачи. Пользователем разработанного на основе каркаса компонента также является разработчик. Пользователем продукта, построенного на основе этого компонента или компонентов, будет уже не разработчик, а конечный пользователь, для которого создавался продукт.

Взаимодействие с пользователем. Назовём внешним субъектом каркаса в его детализированном состоянии (то есть в состоянии, когда каркас подведён под нужды соответствующей задачи) объект типа «Клиент», представляющий собой клиента системы, реализуемой посредством компонента. Объект типа «Клиент» обращается к подсистеме главным образом через вызов операций «Начать игру» и «Сделать ход».

Диаграмма классов каркаса представлена на рисунке 2.

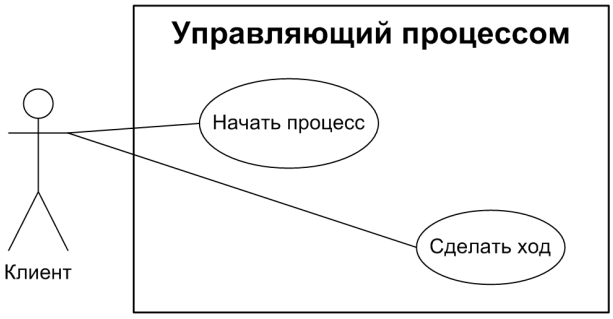

Рис. 1. Диаграмма вариантов использования детализированного каркаса

Структуру каркаса условно можно разделить на два уровня:

1. Уровень основных элементов.
2. Уровень управления процессом.

Элементы уровня основных элементов:

1. «Игрок». Представляет собой главного участника процесса, роль которого может исполнять посредством системы объект типа «Клиент». «Игрок» связан с объектами типов «Соперник» и «Судья».
2. «Соперник». Роль, исполняемая вычислительной машиной, представляет собой противника для объекта типа «Игрок».

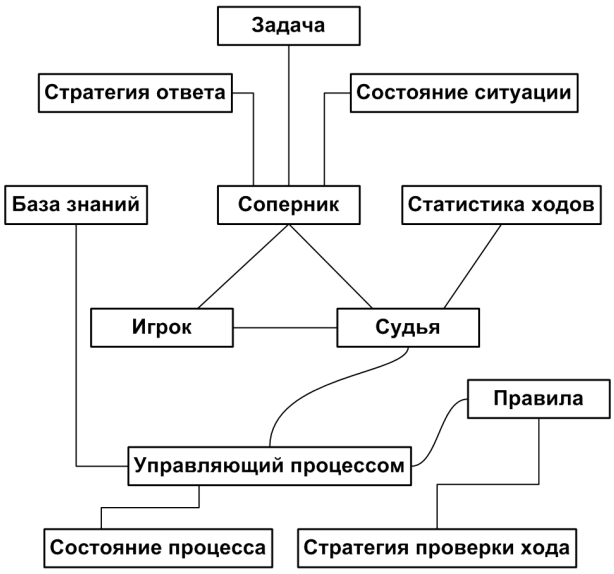

Рис. 2. Диаграмма классов каркаса

3. «Судья». Роль, исполняемая вычислительной машиной, является арбитром игрового процесса.
4. «Задача». Тип, представляющий собой задачу, решаемую во время занятий в игровой форме.
5. «Стратегия ответа». Должна быть задана во время детализации каркаса. В проекте системы может быть реализована посредством шаблона «Стратегия» [6].
6. «Статистика ходов». Тип статистики событий процесса.
7. «Состояние ситуации». Показывает состояние решения задачи. Возможные варианты решения задачи:
   - «Ещё не решена» - означает, что задача ещё не решена.
   - «Почти решена» - некоторые части задачи еще не решены.
   - «Решена» - задача решена полностью.

Элементы уровня управления процессом:

1. «Управляющий процессом». Тип объекта, представляющего сущность, знакомую с элементами процесса и принципами его организации. Содержит в себе описание события «Обработка вопроса», вызываемого действиями объекта типа «Игрок», когда он обменивается сообщениями с объектом типа «Соперник» во время игры. На это событие объект типа «Клиент» должен подписаться до начала процесса.
2. «Правила». Описание правил процесса.
3. «Стратегия проверки хода». Так же, как и стратегия ответа, должна быть задана во время детализации каркаса.
4. «База знаний». Тип, описывающий объекты, способные управлять знаниями.
5. «Состояние процесса». Описывает состояние процесса. Процесс может находиться в двух состояниях:

- «Активный» - процесс в активном состоянии.
- «Неактивный» - процесс находится в неактивном состоянии.

Поток управления. Детализация (или специфицирование под конкретный случай) каркаса происходит следующим образом. Создаётся новый тип (класс в терминах объектно-ориентированной парадигмы), производится операция наследования этим типом структуры и поведения, присущих классу «Управляющий процессом». Задаётся стратегия проверки хода, сделанного игроком, и стратегия ответа или, другими словами, реакция на ход игрока. Помимо этого необходимо создать задачу и присвоить её представление объекту типа «Задача», доступного благодаря наследованию от класса «Управляющий процессом». Диаграмма последовательностей детализации каркаса представлена на рисунке 3.

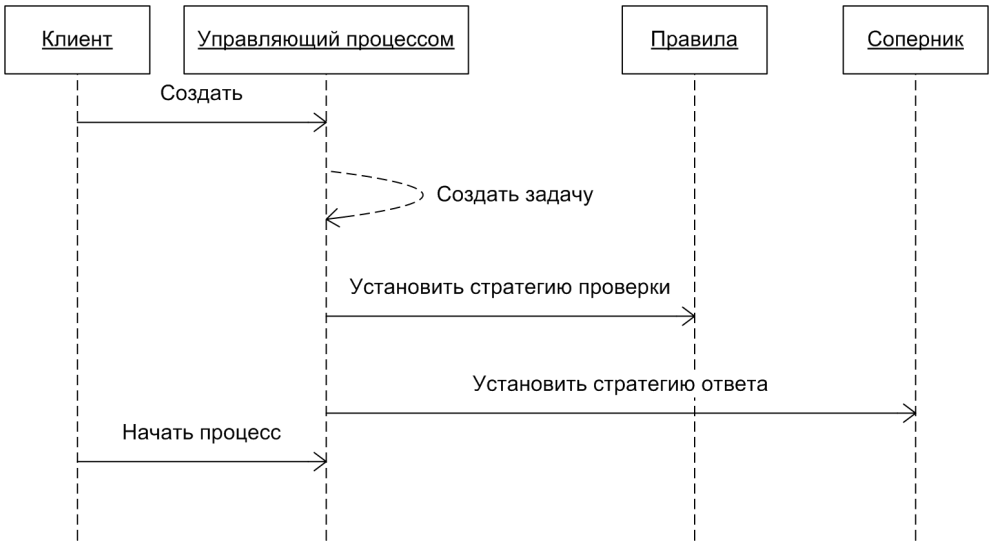

Рис. 3. Диаграмма последовательностей детализации игрового каркаса

Последовательность взаимодействия с детализированным каркасом (рисунки 4 и 5) предполагает следующие действия:

1. Создание экземпляра класса (обязанность объекта типа «Клиент»), наследованного от «Управляющий процессом», который создаёт экземпляры следующих типов:
   - «Игрок»,
   - «Соперник»,
   - «Судья»,
   - «Правила»,
   - «База знаний».
2. Старт процесса.
3. Осуществление игрового хода. После чего происходит событие обработки вопроса, передающее объекту типа «Клиент» информацию о проделанном игровом ходе.

Описание поведения. При выполнении операции «Сделать ход» происходит следующая последовательность действий:

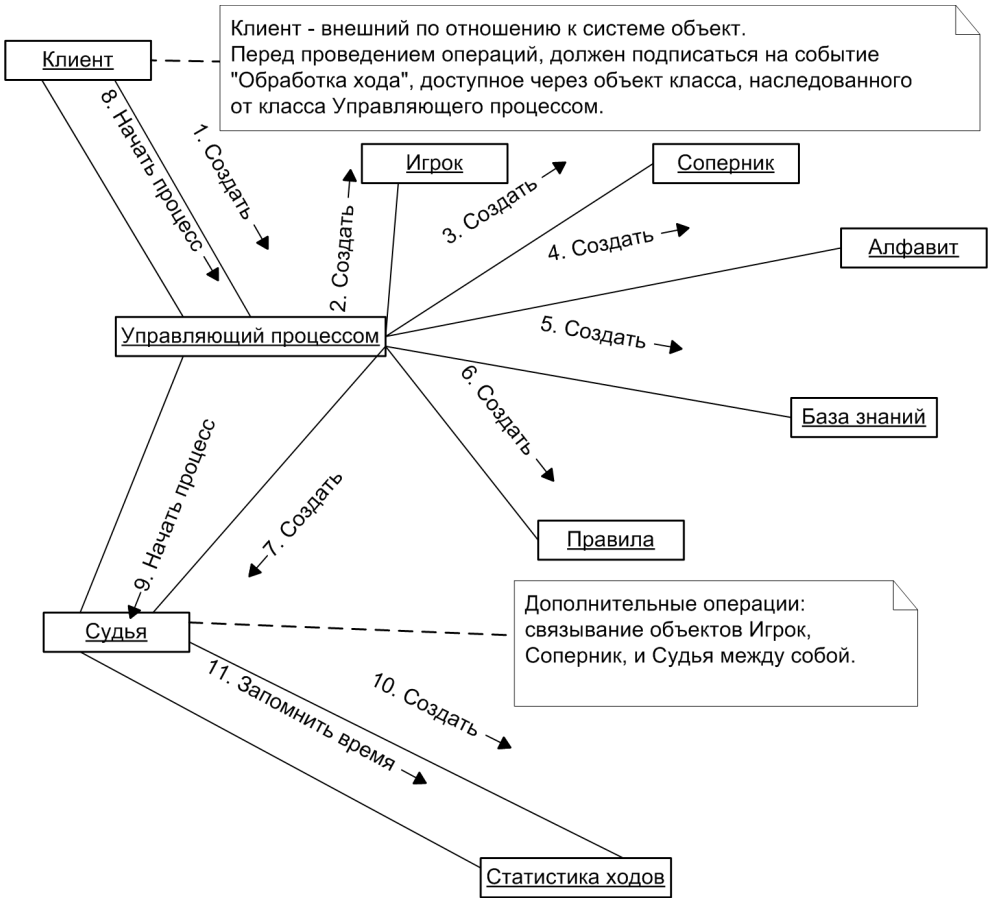

Рис. 4. Диаграмма сотрудничества объектов каркаса во время операции «Начать процесс»

1. После получения в качестве входного параметра списка вопросов осуществляется проверка условия «Процесс находится в активном состоянии». При положительном ответе на утверждение условия происходит переход на следующий шаг, иначе – выход из подпрограммы.
2. Передаётся сообщение объекту типа «Игрок» - «Совершить ход».
3. Происходит проверка состояния задачи, после чего, если состояние изменилось, происходит фиксация этого изменения.
4. Происходит событие обработки вопроса, заданного объектом типа «Игрок» во время хода.

При выполнении объектом типа «Игрок» операции «Совершить ход» происходит следующая последовательность действий:

1. После получения в качестве входных параметров списка вопросов и объекта типа «Правила» производится операция - «Получить ответ от соперника».
2. Обновляется состояние решения задачи.
3. Происходит обновление статистики в объекте типа «Судья».
4. Возвращается состояние задачи.

При выполнении операции «Получить ответ от соперника» происходит сле-

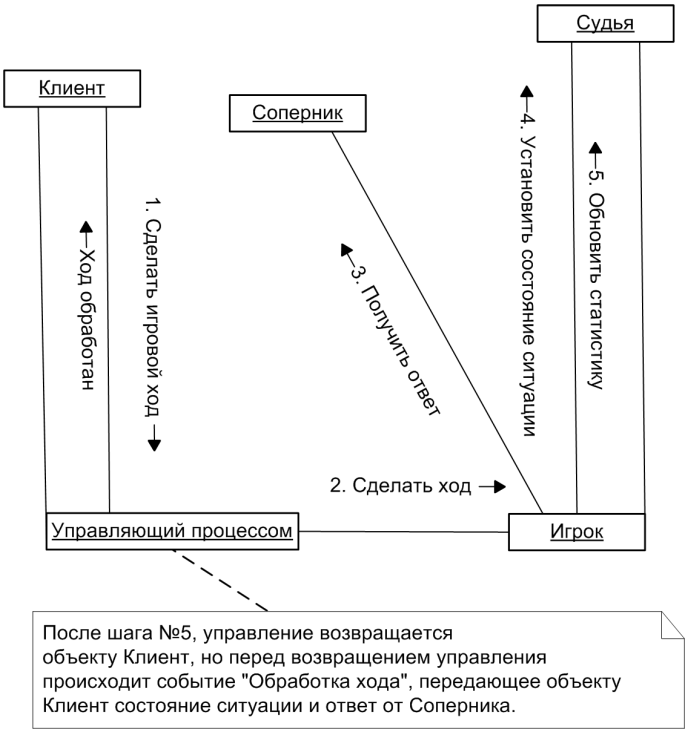

Рис. 5. Диаграмма сотрудничества объектов каркаса во время операции «Сделать игровой ход»

дующая последовательность действий:

1. После получения в качестве входных параметров списка вопросов и правил игры выполняется операция проверки хода в соответствии с правилами.
2. Происходит проверка результата предыдущего действия. При удачной проверке происходит выполнение операции «Получить ответ» согласно заданной стратегии, описанной тем, кто производил детализацию каркаса. В противном случае система вынуждена сообщить о некорректности хода.

При выполнении операции «Получить результат проверки» происходит последовательность действий, приведённых ниже:

1. После получения в качестве входных параметров списка вопросов и объекта типа «Судья» происходит проверка условия «Вопрос задан в первый раз». Если условие выполняется, происходит переход на шаг 2, иначе – на шаг 4.
2. Происходит получение списка вопросов, заданных в процессе игры.
3. Проверяется наличие текущего вопроса в списке. При условии нахождения запрашиваемого элемента возвращается результат «Имело место повторение».
4. Происходит выполнение операции «Проверить вопросы» в соответствии с заданной стратегией, описанной тем, кто производил детализацию каркаса.

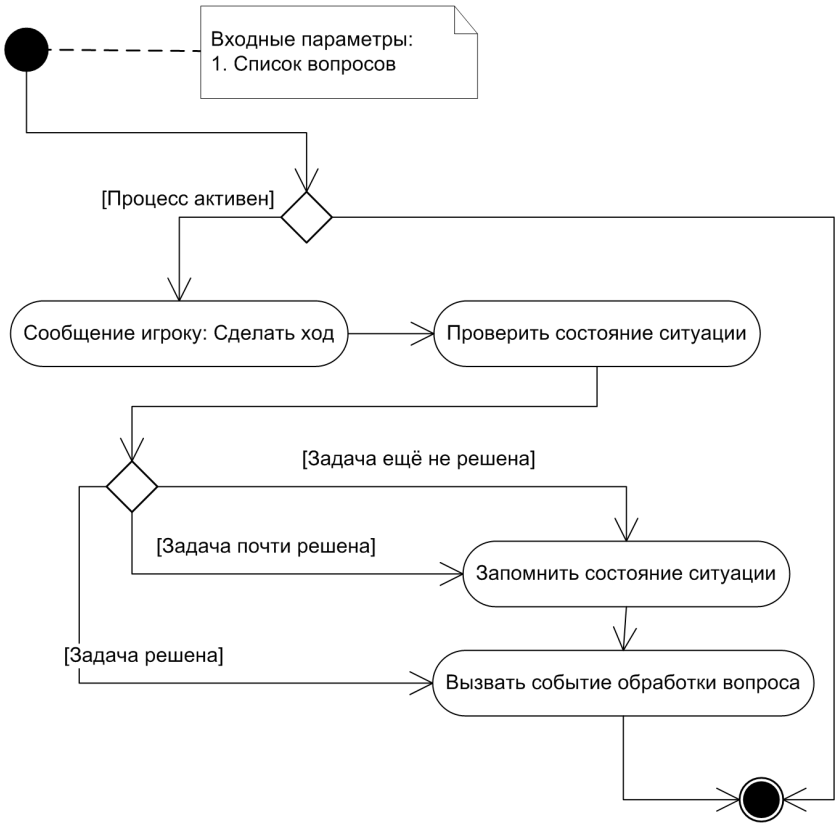

Рис. 6. Диаграмма деятельности, описывающая последовательность действий при вызове операции «Сделать ход»

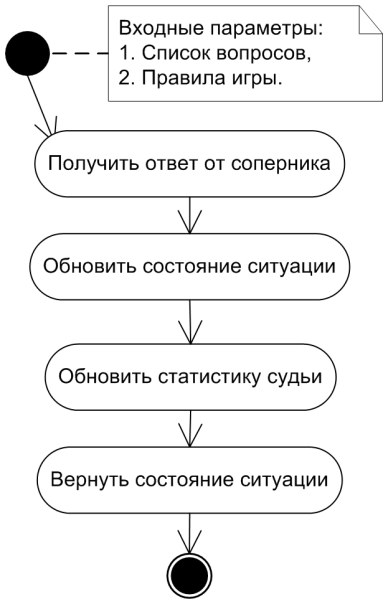

Рис. 7. Диаграмма деятельности, описывающая последовательность операций при выполнении операции «Совершить ход» на объекте типа «Игрок»

Рис. 8. Диаграмма деятельности, описывающая последовательность действий при вызове операции «Получить ответ от соперника»

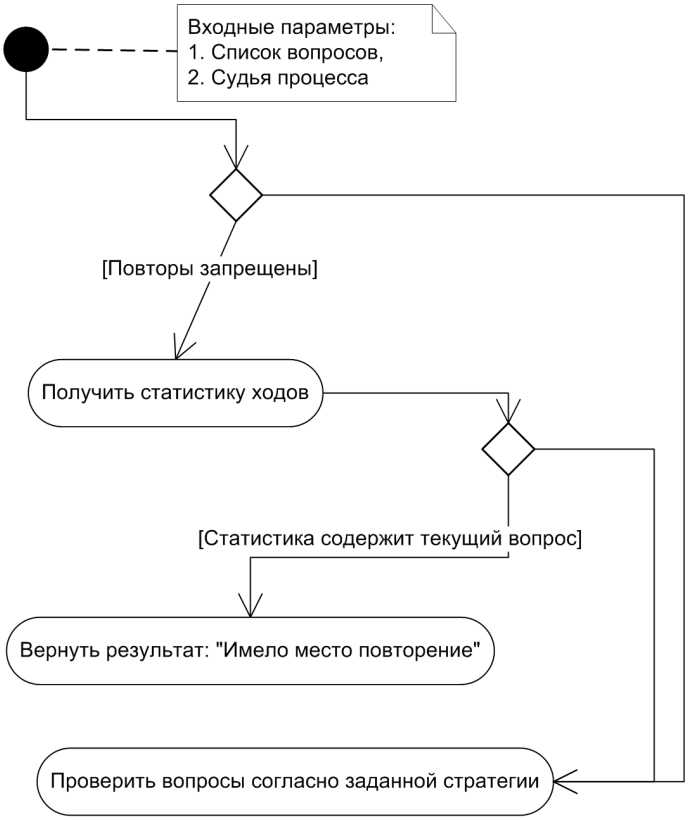

Рис. 9. Диаграмма деятельности, описывающая последовательность операций при вызове метода «Получить результат проверки»

Заключение

В работе была представлена высокоуровневая модель каркаса. Реализация описанной модели способствует упрощению процесса создания программных компонентов поддержки занятий в игровой форме. Модель использовалась автором для создания проекта игровых компонентов. Воплощение этого проекта способствовало реализации готовых компонентов, которые были применены в процессе разработки одного из обучающих игровых приложений.

Литература

1. Parnas, D.L. On the Design and Development of Program Families // IEEE Transactions on Software Engineering, Vol. SE-2, No. 1, March 1976. P. 1–9.
2. Prensky M. Digital Game-Based Learning. U.S.A.: McGraw-Hill, 2004.
3. Рамбо Д., Блаха М. UML 2.0: объектно-ориентированное моделирование и разработка, 2-е издание. СПб.: Питер, 2007.
4. Fowler M. Analysis Patterns: Reusable Object Models. U.S.A.: Addison-Wesley, 1997.
5. Fowler M. Patterns of Enterprise Application Architecture. U.S.A.: Addison-Wesley, 2002.
6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2008.
7. Бек К. Шаблоны реализации корпоративных приложений. М.: ИД Вильямс, 2008.
8. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике, 2-е издание. СПб.: Питер, 2006.
9. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ-Плюс, 2001.