УДК 004.41

ПРОЕКТИРОВАНИЕ СИСТЕМЫ АВТОМАТИЗИРОВАННОЙ ПРОВЕРКИ РЕШЕНИЙ ДЛЯ ПРОВЕДЕНИЯ СОРЕВНОВАНИЙ ПО ПРОГРАММИРОВАНИЮ

© 2013 М.С. Русакова, Р.И. Меерсон[1]

В статье рассматривается проектирование системы автоматизированной проверки решений, предназначенной для проведения региональных соревнований по программированию. В работе выявлены требования и ограничения, накладываемые на системы данного рода. Спроектирована база данных, обеспечивающая хранение задач для соревнований. Для построения системы использованы объектный подход и концепция паттернов. Проектирование системы проведено с использованием унифицированного языка моделирования. Построены UML-диаграммы классов приложения, а также диаграммы последовательности процесса проверки решения.

Ключевые слова: олимпиадное программирование, система автоматизированной проверки решений, паттерны проектирования, объектный подход к проектированию, язык UML, база данных.

Введение

Олимпиады, соревнования (командное и личное первенство) и чемпионаты по информатике и программированию широко распространены и имеют достаточно долгую историю. Командный студенческий чемпионат мира по программированию ACM ICPC (Association for Computing Machinery International Collegiate Programming Contest) проводится с 1977 года. Международная олимпиада школьников по информатике IOI (International Olympiad in Informatics) проводится с 1989 года. Отличительной особенностью подобных соревнований является необходимость за ограниченное количество времени (5 часов) оформить решение набора задач (от 8 до 15 заданий) в виде программы на алгоритмическом языке и отправить решение автоматической проверяющей системе. Проверяющая система сразу выдает вердикт: принято решение, или оно отклонено по какой-либо причине. В современных информационных системах подобного профиля в процессе проведения соревнования автоматически верстается таблица результатов, называемая

[1]Русакова Маргарита Сергеевна (ruma@samsu.ru), Меерсон Роман Ильич (homich1991@gmail.com), кафедра информатики и вычислительной математики Самарского государственного университета, 443011, Российская Федерация, г. Самара, ул. Акад. Павлова, 1.

монитор соревнований, в которой показан рейтинг участников. В мониторе отражена информация о том, кто сдал решение, сколько времени на него затрачено, какая именно задача была решена, сколько было сделано попыток сдать решение, какой "штраф" начислен за неудачные попытки. Победитель соревнований определяется автоматически — им становится участник с максимальным рейтингом (решивший наибольшее количество задач за наименьшее время).

Информационные системы (ИС), позволяющие производить автоматическую проверку решений, разрабатываются, как правило, для каждого организатора соревнований. Связано это со спецификой требований в каждом конкретном случае. Соответственно, все подобные системы обладают своими преимуществами и недостатками, которые выражаются в способах настройки системы, простоте доступа к условиям задач, возможности одновременного проведения нескольких соревнований. В качестве примеров можно привести ИС, разработанные в Санкт-Петербурге, Саратове и Самаре. Все они в основе своей схожи, абсолютно функциональны и позволяют проводить соревнования в полном объеме. Однако в саратовской системе [1] нельзя организовать одновременно несколько соревнований. Система СПбНИУ ИТМО — PCMS2 [2] требует сложных конфигурационных файлов и скриптов запуска даже для создания простого соревнования. Система СамГУ [3] имеет некоторые сложности, связанные с проверкой решений, написанных на Java (для адекватной проверки решения на код налагается ряд условий на расстановку операторных скобок, имя основного класса и т. п.). Проектирование и построение таких ИС являются нетривиальной задачей, т. к. единого подхода к к автоматизации проверки решений нет.

В данной работе рассматривается проектирование автоматизированной системы проверки решений, обладающей всеми функциональными возможностями вышеупомянутых аналогов и позволяющей проводить параллельные соревнования. Система должна быть более простой в запуске и легкой в настройке, чем ее аналог PCMS2 [2]. Кроме того, в рассматриваемой системе должны быть сняты ограничения на текст кода решения (в смысле свободы способа именования основного java-класса и его методов, количества пустых строк в начале и конце программы и т. п.).

1. Описание предметной области и определение требований к информационной системе

Любое соревнование, проводимое с использованием автоматической проверяющей системы, требует тщательной подготовки. Очевидно, система должна поддерживать возможность проверки решений, написанных на разных алгоритмических языках программирования, т. е. в системе должна быть предусмотрена возможность подключения необходимого набора поддерживаемых компиляторов. Согласно стандарту ACM [4], ИС должна поддерживать компиляторы C, C++, Java. В рамках других соревнований набор языков и компиляторов может быть расширен. Основным элементом соревнования является пакет задач, каждая из которых должна быть однозначно идентифицирована (в рамках соревнования — это номер задачи, обозначаемый буквами русского или латинского алфавита). Задача как минимум должна содержать название, "легенду", условие, форматы входных и выходных данных [5]. Кроме того, для каждой задачи и каждого поддерживаемого языка программирования устанавливаются ограничения на максимальное время,

за которое написанная программа должна выдать ответ, и на максимальное количество используемой программой-решением оперативной памяти.

Для проверки правильности решений в системе необходимо хранить набор тестов, составляемых для каждой задачи, либо проверяющую программу (чекер). В общем случае один тест представляет собой набор из двух текстовых файлов — входного и выходного, составленных в соответствии с форматом данных из условия задачи. Во входном (input) файле приведены данные, передаваемые программе, присланной на проверку. В выходном (output) файле приведены данные, с которыми сравнивается результат работы проверяемой программы. Если задача имеет несколько правильных вариантов выходных данных на одни и те же входные условия, тест содержит только входной файл. Удовлетворяет ли результат работы программы условию задачи, проверяется при помощи чекера. При формировании отклика (вердикта) системы на результат проверки каждого теста надо учесть следующие ситуации: на данном тесте получен верный ответ (Accepted), получен неверный ответ (Wrong Answer), превышен лимит времени для получения ответа (Time limit Exceeded), превышен максимальный возможный объем памяти для данного теста (Memory limit Exceeded), превышено время бездействия (Idleness limit Exceeded), произошла ошибка времени выполнения (Runtime Error), произошла ошибка компиляции (Compilation Error). Задача считается сданной (участник получает отклик системы Accepted), если все тесты пройдены успешно. В любых других случаях система должна отправить участнику свой вердикт с указанием номера теста, где произошел сбой, и количество штрафных очков за неудачную попытку (обычно такие попытки караются 20 минутами штрафного времени).

Основываясь на приведенном описании, выделим основные функциональные требования к системе. Система должна:

- поддерживать параллельное проведение нескольких соревнований;
- в рамках соревнования автоматически проверять присылаемые зарегистрированными участниками решения и поддерживать с ними обратную связь, выдавая сообщение о вердикте ИС;
- автоматически проверять решения на разных языках программирования (поддерживать несколько разных компиляторов с возможностью добавления новых);
- автоматически формировать таблицу текущих результатов, при необходимости "замораживать" монитор для сохранения интриги соревнования;
- сохранять в базе данных сведения о соревнованиях, участниках и задачах, а также ограничениях на задачи, тестах и чекерах;
- для зарегистрированного участника (которым может быть как один человек, так и команда из трех человек) вести "историю" и рейтинг сданных им решений.

Среди нефункциональных требований к разрабатываемой системе можно отметить следующие.

- Требования к реализации. Система должна:
  - быть реализована на языке Java в виде клиент-серверного приложения с графическим интерфейсом пользователя;

  - обеспечивать легкую и удобную настройку и сохранять успешные настройки в виде динамически подгружаемых XML-файлов.

- Требования к интерфейсу:
  - должно быть предусмотрено разделение прав доступа администратора системы (организатора соревнований) и зарегистрированного участника;
  - следует обеспечить как русскоязычный, так и англоязычный интерфейс пользователя.

- Требования надежности: при отключении сервера во время соревнований пользователь должен быть выведен на окно входа в систему, монитор соревнований должен быть сохранен, а время перезапуска сервера не должно превышать нескольких минут.

- Требование к безопасности: необходимо обеспечить сохранность передаваемых данных (шифрование отправляемого на проверку кода).

- Требования к совместимости: сервер должен работать в ОС семейства Windows XP и выше, клиентское приложение должно быть кроссплатформенным. Для успешного функционирования системы необходима виртуальная java-машина версии 1.7 или выше.

Анализируя выдвинутые к системе требования, принято решение, что информация о соревнованиях и участниках, задачи и тесты должны храниться в базе данных (БД), а функциональные возможности ИС должны быть реализованы в клиент-серверном приложении.

2. Проектирование базы данных

Остановимся подробнее на проектировании базы данных. Из семантического описания процесса подготовки и проведения соревнования можно выделить следующие основные сущности.

- "Соревнование" — хранит необходимую информацию о соревновании: уникальный идентификатор, название соревнования, дату и время начала, дату и время окончания, дату и время заморозки монитора.

- "Участник" — содержит информацию о персоне или команде, включая сведения для доступа участника в систему: идентификатор участника, логин, пароль, информацию об участнике (строка информации для отображения в системном мониторе, где могут быть перечислены учебное заведение, фамилии участников команды и т. п.), администратор (атрибут, показывающий наличие администраторских прав).

- "Задача" — хранит информацию об условии, наборе тестов и ограничениях на решение задачи: идентификатор, название задачи, путь к директории с тестами (либо чекерами), количество тестов, ограничение по времени, ограничение по памяти.

- "Решение" — содержит информацию о присланном на проверку решении: уникальный идентификатор; используемый язык программирования; максимальное время, потраченное программой на прохождение каждого теста; максимальный размер памяти, затраченной на прохождение всех тестов; вердикт; время отправки решения.

В информационной модели БД между сущностями "Соревнование" и "Участник" определена связь многие-ко-многим, т. к. в одном соревновании принимают участие много участников (в реальности, на региональном уровне в среднем соревнуется по 25–50 команд), а один и тот же участник может играть в нескольких соревнованиях. В общем случае такой же вид связи поддерживается между "Задачей" и "Соревнованием". Связи сущности "Решение" с сущностями "Соревнование", "Участник" и "Задача" — многие-к-одному. Реляционная модель, полученная после процедуры нормализации, приведена к третьей нормальной форме и представлена на рис. 1. Связи вида многие-ко-многим выведены из модели добавлением вспомогательных таблиц. Кроме того, для удобства добавлены справочники "Вердикт", "Язык" и "Ограничения". Для реализации базы данных выбрана СУБД MySQL 5.5.

3. Проектирование приложения. Серверная часть

Исходя из выдвинутых требований к системе, было принято решение о создании клиент-серверного приложения, причем на стороне сервера должны осуществляться следующие функции:

- запуск нескольких параллельных соревнований;
- поддержка одновременного подключения многих (в рамках реальных соревнований — нескольких десятков) пользователей;
- автоматическая проверка решения и отправка вердикта участнику;
- подключение к базе данных, отправка запросов и получение ответов от БД.

Так как вся основная функциональность по тестированию решений и доступу к БД переложена на серверное приложение, то мы имеем дело с моделью "толстый сервер — тонкий клиент".

Остановимся подробнее на некоторых важных моментах проектирования серверной части приложения. Проектирование рассматриваемой системы проводилось с использованием унифицированного языка моделирования [6]. Была выбрана объектная модель приложения, и выделены основные классы, приведенные на UML-диаграмме (рис. 2), которые затем должны быть реализованы на платформе Java 1.7 в среде разработки NetBeans IDE 7.1.1.

"Сервер" — основной класс, в котором хранится список подключенных в данный момент участников, ссылка на подключение к базе, а также диспетчер подключений к серверу. В классе описаны методы запуска и остановки сервера, отправки сообщений всем пользователям и конкретному пользователю. "Сервер" реализует интерфейсы "Обработчик сообщений" и "Слушатель соединения". Интерфейс "Обработчик сообщений" содержит метод обработки сообщений разных типов. Интерфейс "Слушатель соединения" позволяет серверу реализовать шаблон проектирования "Слушатель" [7] и позволяет запускать процесс обработки полученного

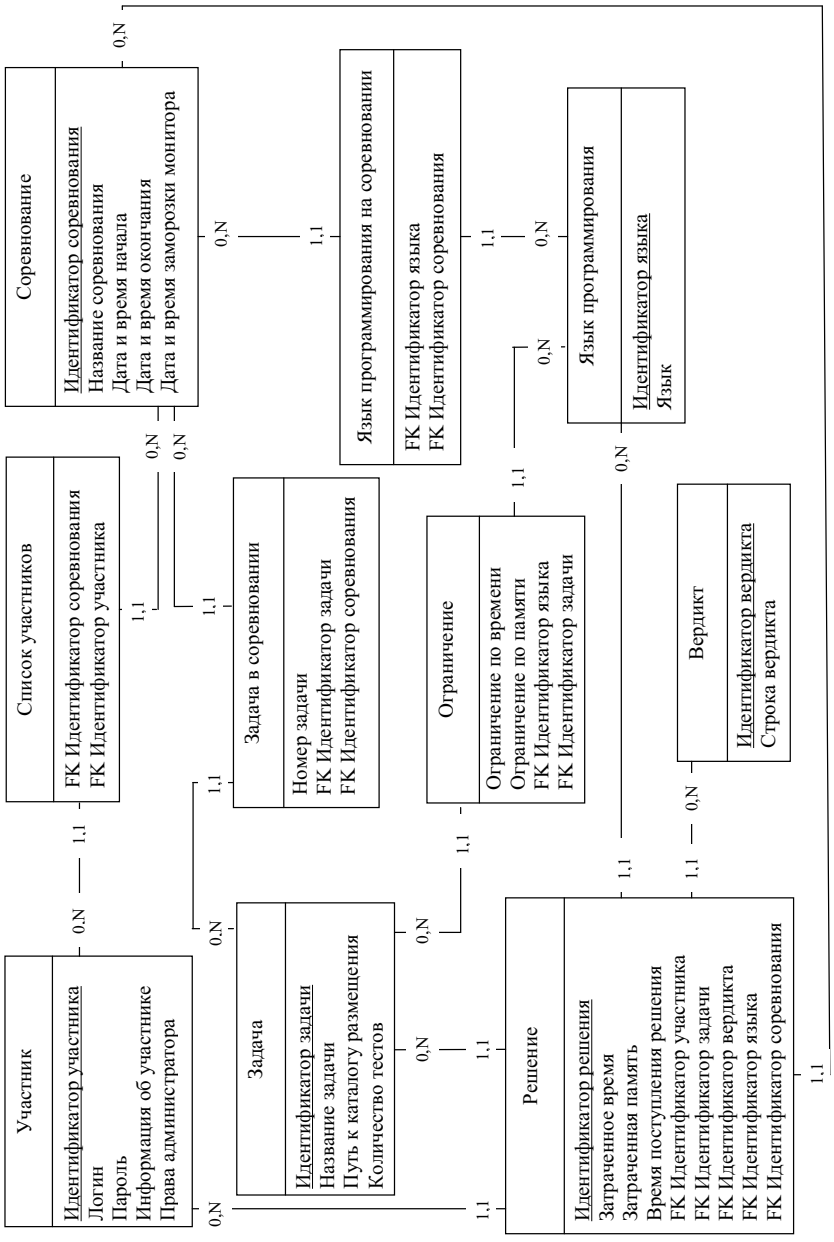

Рис. 1. Реляционная модель базы данных

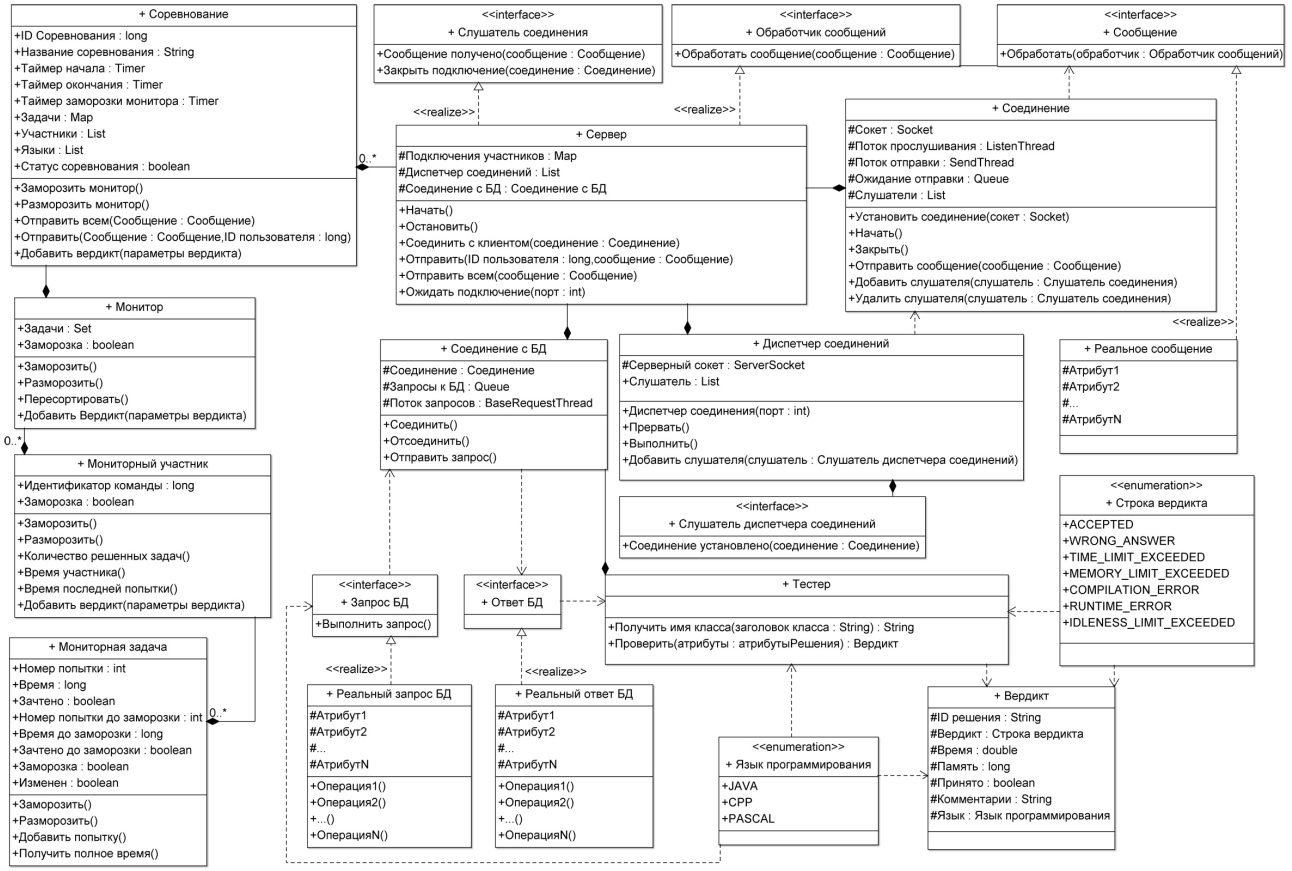

Рис. 2. Диаграмма классов серверной части приложения

сообщения. "Сервер" поддерживает связь по типу агрегации с классом "Диспетчер соединений", который позволяет подключать к серверу новых клиентов и устанавливать с ними связь. Для каждого клиента на сервере создается объект класса "Соединение", где содержится информация о сокете подключения клиента, потоки прослушивания сообщений от клиента и отправки ему сообщений. В экземпляре класса "Соединение" формируется очередь сообщений. Поток отправки сообщений клиенту периодически просматривает очередь, и, если она не пуста, происходит отправка сообщений клиенту. Были созданы также служебные классы "Язык программирования" и "Строка вердикта", в которых соответственно перечислены доступные в проверяющей системе языки программирования и возможные ответы системы.

Для проверки (тестирования) присланного решения на сервере создается экземпляр класса "Тестер" и запускается его метод "Проверить()", в который передаются ID пользователя (автора решения), ID решения, код (текст) присланного решения, язык программирования, ID задачи, ограничения задачи. Этот метод запускает проверку решения и возвращает экземпляр класса "Вердикт", который содержит вердикт решения и дополнительную информацию. Если же решение выполнено на языке Java, то запускается метод класса "Тестер", выявляющий имя открытого класса в решении (так называемый метод-парсер, снимающий ограничения на именование Java-классов и способы расстановки операторных скобок).

Также на сервере хранится список классов "Соревнование", в экземплярах которых содержится информация о запланированных соревнованиях. При запуске сервера загружается информация из базы данных о предстоящих соревнованиях. На каждое соревнование создается отдельный экземпляр класса "Соревнование". В каждом подобном объекте хранятся таймеры запуска и остановки соревнования, а также таймер заморозки монитора. Также экземпляр класса содержит список участников и монитор соревнования (как объект класса "Монитор"), содержащий отсортированное множество классов "Мониторный участник". Класс хранит информацию обо всех вердиктах по задачам определенного участника при помощи объектов типа "Мониторная задача". В классе "Мониторный участник" реализованы методы добавления вердикта по конкретной задаче, подсчета суммарного времени и числа полностью решенных задач. Класс "Мониторная задача" хранит информацию о конкретной задаче (номер попытки, время последней попытки, решена задача или нет). В классе реализована функция подсчета штрафного времени по задаче.

При проведении объектного проектирования серверного и клиентского приложений оказалось удобным воспользоваться концепцией паттернов (или образцов) [7; 8]. При проектировании механизма обращения к базе данных с сервера был использован шаблон проектирования "Команда". В системе предусмотрены интерфейсы: "Запрос БД" и "Ответ БД". Интерфейс "Запрос БД" содержит метод, который производит запрос к БД. Также существует класс подключения к базе "Соединение с БД", где содержится очередь запросов. При создании экземпляра данного класса формируется поток, проверяющий, не пуста ли очередь запросов. Если она не пуста, поток запускает метод "Выполнить запрос()" первого запроса из очереди, после чего запрос из очереди удаляется. Если какому-то потоку необходимо получить данные из базы, то ему необходимо добавить свой запрос в очередь и ожидать ответа от базы. Такой механизм позволяет избежать коллизий, возникающих, например, при одновременных запросах к базе на чтение и запись данных.

Остановимся подробнее на проектировании и реализации механизма автоматической проверки решений. Пользователь отправляет из клиентского приложения сообщение с решением задачи на сервер. Серверное приложение генерирует поток тестирования, в котором формируется запрос к БД на получение ограничений и тестов для проверяемой задачи. База данных возвращает серверному приложению ответ, после чего собственно запускается тестирование (проверка). По окончании проверки системой генерируется вердикт, который возвращается в поток тестирования. Далее вердикт записывается в базу и отправляется пользователю. Генерация отдельного потока тестирования для каждого решения, присылаемого на сервер, позволяет обеспечить распараллеливание процесса проверки решений, чем значительно его ускоряет.

Рис. 3. Диаграмма последовательности для механизма тестирования задачи проверяющей системой

Рассмотрим сам процесс тестирования более детально (рис. 3). После запуска проверки решений в заранее определенной директории создается папка для тестирования решения. Имя папки уникально, т. к. оно эквивалентно ID решения. Далее формируется файл с кодом решения, который отправляется на компиляцию, и, если она прошла успешно, скомпилированный файл сохраняется в папке тестирования. Если компиляции закончилась неудачей, то возвращается соответствующий вердикт, иначе тестирование продолжается. В папку добавляются тесты, и решение последовательно проверяется на них, начиная с первого. Если на каком-то тесте решение не получило вердикт Accepted, то проверка останавливается и возвращается вердикт. Если решение получило Accepted на всех тестах, пользователю отправляется сообщение об успешной сдаче решения. По окончании процесса тестирования папка удаляется. Все шаги процесса тестирования автома-

тически производятся проверяющей системой и не требуют какого-либо участия пользователя (кроме отправки решения на проверку).

4. Клиентская часть приложения

Клиентское приложение проверяющей системы спроектировано с учетом разделения ролей администратора и участника соревнований (рис. 4). Интерфейс пользователя для роли "участник соревнований" дает возможность зарегистрироваться в системе, прочитать условия задач в текущем соревновании, отправить решение на проверку и просмотреть системный монитор. Администратор системы, обладая всеми возможностями обычного пользователя, также имеет право создавать соревнования, задачи, добавлять условия задач и тесты для проверки решения, просматривать присланные решения и при необходимости наделять других пользователей правами администратора.

Рис. 4. Диаграмма вариантов использования

Остановимся подробнее на проектировании объектной структуры клиентского приложения. Рассмотрим механизм передачи сообщений между сервером и клиентским приложением. Отметим, что для подключения к серверу и обмена сообщениями с ним клиентское приложение использует структуру классов (рис. 5), похожую на фрагмент диаграммы классов сервера. Основным классом приложения является "Клиент", содержащий информацию о подключении к серверу и ID участника; в нем реализованы методы подключения/отключения от сервера. "Клиент" реализует интерфейсы "Диспетчер сообщений" и "Слушатель соединения". Для "общения" клиента и сервера разработан механизм передачи сообщений на базе шаблона проектирования "Посетитель". На сервере и клиенте существуют объекты класса "Соединение", предназначенные для отправки и приема сообщений. Все сообщения реализуют интерфейс "Сообщение", в котором существует единственный метод "Обработать сообщение()". Сервер и клиент реализуют интерфейс "Диспетчер сообщений", в котором предусмотрен метод обработки всех типов сообщений. Отправитель генерирует сообщение и передает его своему экземпляру класса "Соединение". Последний сериализует сообщение и передает его на сторону получателя. Экземпляр класса "Соединение" на стороне получателя десериализует сообщение и запускает метод "Обработать сообщение()", куда передает получателя

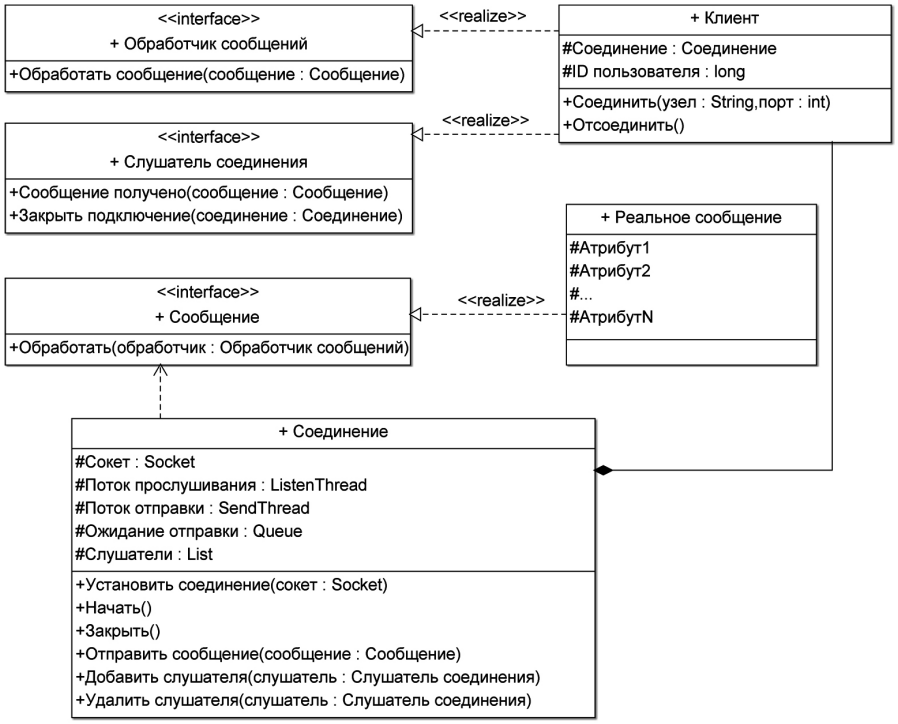

Рис. 5. Диаграмма классов клиентского приложения

как параметр. Метод вызывает у получателя требуемую операцию обработки сообщения, куда и передает само сообщение. Таким образом, используя концепцию паттернов проектирования [7; 8], мы получили легко масштабируемую систему приема и отправки сообщений произвольных типов.

Заключение

Таким образом, в настоящей работе на основе объектного подхода и концепции паттернов спроектирована информационная система для проведения соревнований по программированию, обеспечивающая автоматическую проверку присылаемых участниками решений. Среди несомненных плюсов разработанной ИС можно отметить простоту в настройке и использовании, кроссплатформенность (требуется только java-машина, операционная система может быть любой), масштабируемость (в систему можно без затруднений добавить новые типы сообщений и запросов к базе данных, поскольку эти механизмы организованы при помощи шаблонов проектирования).

Перспективы дальнейшего развития и применения разработанной проверяющей системы весьма обширны. Система может быть использована собственно по назначению: для проведения соревнований и тренировок участников олимпиад и чемпионатов по программированию. ИС может также успешно применяться в учебном процессе на механико-математическом факультете для проведения зачетных, итоговых и промежуточных контрольных занятий по языкам программирова-

ния, дисциплинам специализации и вычислительным практикумам, структурам и алгоритмам кодирования данных и др. Очевидно, что для применения в учебном процессе ИС легко может быть модифицирована от подсчета штрафного времени и количества попыток сдать задачу к подсчету баллов.

Литература

[1] Сайт соревнований по спортивному программированию Саратовского государственного университета (ACM International Collegiate Programming Contest/Southern Subregional Contest). URL: http://contest.sgu.ru.

[2] Система PCMS2 проведения соревнований по спортивному программированию Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики (North-Eastern European Regional Contest). URL: http://neerc.ifmo.ru/uc/pcms2-v2.html.

[3] Сайт соревнований по спортивному программированию Самарского государственного университета. URL: http://contest.uni-smr.ac.ru.

[4] Правила проведения региональных соревнований по стандарту ACM (ICPC Regional Rules). URL: http://icpc.baylor.edu/info/Regional+Rules.

[5] Оршанский С. О решении олимпиадных задач по программированию формата ACM ICPC // Информатика. 2006. №1. URL: http://inf.1september.ru/article.php?ID=200600106.

[6] Фаулер М. UML. Основы. СПб.: Символ, 2005. 184 с.

[7] Фримен Э. Паттерны проектирования. СПб.: Питер, 2011. 645 с.

[8] Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. СПб.: Питер, 2001. 368 с.

Поступила в редакцию 1/*II*/2013;
в окончательном варианте — 1/*III*/2013.

DESIGN OF AUTOMATED SOLUTION CHECK SYSTEM FOR PROGRAMMING CONTESTS

© 2013 M.S. Rusakova, R.I. Meerson[2]

The paper deals with design of automated solution check system for regional programming contests. The requirements and constraints for such systems are found in the work. Database for contest tasks is designed. The system is constructed using object modeling and patterns concept. Unified modeling language is used for system design. UML-diagrams are shown for project classes as well as diagram for solution check process.

Key words: contest programming, automated checking system, design patterns, object-oriented designing, using UML, database.

Paper received 1/*II*/2013.
Paper accepted 1/*II*/2013.

[2]Rusakova Margarita Sergeevna (ruma@samsu.ru), Meerson Roman Iliych (homich1991@gmail.com), the Dept. of Informatics and Computing Mathematics, Samara State University, Samara, 443011, Russian Federation.